
THE UNIVERSITY
*of* EDINBURGH

# Dynamic Partial Reconfiguration Management for High Performance and Reliability in FPGAs

by

**Ali Ebrahim**


A thesis submitted in partial fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

The University of Edinburgh

March, 2015

# Declaration

I hereby declare that this thesis was composed and originated entirely by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualifications.

Ali Ebrahim

# Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Professor Tughrul Arslan. His excellent supervision and vast knowledge, have added considerably to my research work. I deeply appreciate his kindness, patience and continuous guidance throughout my graduate experience.

I would like also to express my gratitude to my supervisor, Dr. Khaled Benkrid, who introduced me to the exciting field of reconfigurable computing. I would like to thank him for his guidance and support during the first year of my PhD study. I am also very grateful to Dr. Ahmet Erdogan, who assisted me in all the academic procedures and helped me familiarise myself with the facilities and the labs at the University of Edinburgh.

I am very grateful to the University of Bahrain for providing me with the opportunity to conduct my research work in the UK. I would like to thank all my colleagues at the University of Bahrain for their support and encouragements.

I would like to thank all members of the System Level Integration Group (SLIG) at the University of Edinburgh. My special thanks go to my close colleagues in the group who worked in the R3TOS project. I am very grateful to my senior colleague Dr. Xabier Iturbe, who originally conducted this research project and helped me with his valuable suggestions and guidance. I also acknowledge the contributions of Dr. Chaun Hong, Dr. Hana Hussain and Mr. Jalal Khalifat, who have been closely involved in my research work.

Last but not least, I would like to express my deepest gratitude to all members of my family. Special thanks go to my wonderful parents for their love and support. I must thank and acknowledge my wife and best friend, Bayan, who has patiently supported me during the difficult moments of my research. I also would like to thank my new-born son, Hasan, who gave me the strength and inspiration to complete my thesis.

# Lay Summary of Thesis

Modern Field-Programmable Gate Arrays (FPGAs) have evolved to devices consisting of a large number of reconfigurable hardware resources. Hardware Description Language (HDL) codes can be synthesised to binary files which can be loaded into the FPGA to implement different hardware functions. Similar to traditional software solutions, functions can be easily modified and upgraded using the same hardware components, however, functions implemented on FPGAs can be customised to perform parallel computations and hence achieve a much higher performance compared to software solutions that execute algorithms sequentially.

Recent FPGAs support Dynamic Partial Reconfiguration (DPR) which further enhances the flexibility of the device. DPR allows for changing the functionality of certain blocks within the FPGA while the rest of the FPGA is operational. This means that computational functions can be swapped in and out of the FPGA at run-time while other functions are actively performing computations. DPR has opened the doors to implement highly adaptive systems which can be deployed in many applications. Researchers are continually looking for ways to enhance the reliability and performance of systems implemented in FPGAs using DPR. Enhancing reliability is particularly important in space and militarily electronic systems where FPGAs are usually seen as unreliable devices due to their sensitivity to radiation and harsh environmental conditions.

This thesis proposes a number of novel techniques that deploy DPR to enhance the reliability and performance of reconfigurable systems. A comprehensive internal configuration management system for partially reconfigurable FPGAs is introduced. This system supports high-performance configuration via the Internal Configuration Access Port (ICAP) as well as several parameterisable fault-detection and fault-recovery capabilities.

# Abstract

Modern Field-Programmable Gate Arrays (FPGAs) are no longer used to implement small “glue logic” circuitries. The high-density of reconfigurable logic resources in today’s FPGAs enable the implementation of large systems in a single chip. FPGAs are highly flexible devices; their functionality can be altered by simply loading a new binary file in their configuration memory. While the flexibility of FPGAs is comparable to General-Purpose Processors (GPPs), in the sense that different functions can be performed using the same hardware, the performance gain that can be achieved using FPGAs can be orders of magnitudes higher as FPGAs offer the ability for customisation of parallel computational architectures.

Dynamic Partial Reconfiguration (DPR) allows for changing the functionality of certain blocks on the chip while the rest of the FPGA is operational. DPR has sparked the interest of researchers to explore new computational platforms where computational tasks are off-loaded from a main CPU to be executed using dedicated reconfigurable hardware accelerators configured on demand at run-time. By having a battery of custom accelerators which can be swapped in and out of the FPGA at run-time, a higher computational density can be achieved compared to static systems where the accelerators are bound to fixed locations within the chip. Furthermore, the ability of relocating these accelerators across several locations on the chip allows for the implementation of adaptive systems which can mitigate emerging faults in the FPGA chip when operating in harsh environments. By porting the appropriate fault mitigation techniques in such computational platforms, the advantages of FPGAs can be harnessed in different applications in space and military electronics where FPGAs are usually seen as unreliable devices due to their sensitivity to radiation and extreme environmental conditions.

In light of the above, this thesis investigates the deployment of DPR as: 1) a method for enhancing performance by efficient exploitation of the FPGA resources, and 2) a method for enhancing the reliability of systems intended to operate in harsh environments. Achieving optimal performance in such systems requires an efficient internal configuration management system to manage the reconfiguration and

execution of the reconfigurable modules in the FPGA. In addition, the system needs to support "fault-resilience" features by integrating parameterisable fault detection and recovery capabilities to meet the reliability standard of fault-tolerant applications. This thesis addresses all the design and implementation aspects of an Internal Configuration Manger (ICM) which supports a novel bitstream relocation model to enable the placement of relocatable accelerators across several locations on the FPGA chip. In addition to supporting all the configuration capabilities required to implement a Reconfigurable Operating System (ROS), the proposed ICM also supports the novel multiple-clone configuration technique which allows for cloning several instances of the same hardware accelerator at the same time resulting in much shorter configuration time compared to traditional configuration techniques. A fault-tolerant (FT) version of the proposed ICM which supports a comprehensive fault-recovery scheme is also introduced in this thesis. The proposed FT-ICM is designed with a much smaller area footprint compared to Triple Modular Redundancy (TMR) hardening techniques while keeping a comparable level of fault-resilience.

The capabilities of the proposed ICM system are demonstrated with two novel applications. The first application demonstrates a proof-of-concept reliable FPGA server solution used for executing encryption/decryption queries. The proposed server deploys bitstream relocation and modular redundancy to mitigate both permanent and transient faults in the device. It also deploys a novel Built-In Self-Test (BIST) diagnosis scheme, specifically designed to detect emerging permanent faults in the system at run-time. The second application is a data mining application where DPR is used to increase the computational density of a system used to implement the Frequent Itemset Mining (FIM) problem.

# Related Publications

1. **A Fast and Scalable FPGA Damage Diagnostic Service for R3TOS Using BIST Cloning Technique**
   **Ebrahim. A**, Arslan. T, Iturbe. X
   *The International Conference on Field Programmable Logic and Applications (FPL), pp. 1-4, 2014*

2. **On Enhancing the Reliability of Internal Configuration Controllers in FPGAs**
   **Ebrahim. A**, Arslan. T, Iturbe. X
   *The NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pp. 83-88, 2014*

3. **A Platform for Secure IP Integration in Xilinx Virtex FPGAs**
   **Ebrahim. A**, Benkrid. K, Khalifat. J, Hong. C
   *The International Conference on Reconfigurable Computing and FPGAs (ReConFig), pp. 1-6, 2013*

4. **Multiple-Clone Configuration of Relocatable Partial Bitstreams in Xilinx Virtex FPGAs**
   **Ebrahim. A**, Benkrid. K, Iturbe. X, Hong. C
   *The NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pp. 178-183, 2013*

5. **A Novel High-Performance Fault-Tolerant ICAP Controller**
   **Ebrahim. A**, Benkrid. K, Iturbe. X, Hong. C
   *The NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pp. 259-263, 2012*

# Contents

**Declaration** ..... **I**
**Acknowledgements** ..... **II**
**Lay Summary of Thesis** ..... **III**
**Abstract** ..... **IV**
**Related Publications** ..... **VI**
**Contents** ..... **VII**
**List of Figures** ..... **XI**
**List of Tables** ..... **XIV**
**List of Algorithms** ..... **XVI**
**List of Acronyms and Abbreviations** ..... **XVII**
**Chapter 1 : Introduction** ..... **1**
1.1 Thesis Objectives ..... 5
1.2 Novelty and Contribution ..... 7
1.3 Thesis Outline ..... 9
**Chapter 2 : Introduction to FPGAs and Dynamic Partial Reconfiguration** ..... **12**
2.1 Xilinx FPGAs and Design Flow ..... 13
2.1.1 Overview of Xilinx Reconfigurable Resources ..... 14
2.1.2 Basic Routing and Clocking Structure ..... 17
2.1.3 Basic Design Flow ..... 20
2.2 Dynamic Partial Reconfiguration ..... 22
2.2.1 Xilinx DPR Flow ..... 23
2.2.2 Altera DPR Flow ..... 26
2.2.3 Configuration Ports ..... 28
2.2.4 Bitstream Relocation ..... 28
2.3 Chapter Conclusion ..... 34
**Chapter 3 : Dynamic Partial Reconfiguration for High Performance and Reliability** ..... **36**
3.1 DPR Deployment in High-Performance Systems ..... 37
3.1.1 FPGA-based Acceleration in HPC ..... 37
3.1.2 Reconfigurable Operating Systems ..... 42
3.1.3 Reducing Reconfiguration Delay ..... 45
3.2 DPR for Enhanced Fault-Tolerance ..... 50

3.2.1 Background on Faults in SRAM-FPGAs ..... 51
3.2.2 Reliability Features in Modern FPGAs ..... 53
3.2.3 DPR Techniques for Enhanced Fault-Tolerance ..... 54
3.3 Chapter Conclusion ..... 61
**Chapter 4 : A High-Performance Internal Configuration Manager ..... 63**
4.1 General Architecture of the ICM ..... 64
4.1.1 Building Blocks of the ICM ..... 64
4.1.2 Interfacing with the Main CPU ..... 65
4.1.3 The Configuration Operations ..... 66
4.2 The ICAP Controller ..... 68
4.2.1 Basic Operation of the Internal Configuration Access Port ..... 70
4.2.2 Fast Operation Set-up ..... 72
4.2.3 The Data-Transfer Phase ..... 77
4.2.4 The Configuration Verification Phase ..... 86
4.3 The External Memory Controller ..... 86
4.4 Multiple-Clone Configuration ..... 88
4.4.1 Overview ..... 90
4.4.2 The Clonable Partial Bitstream ..... 91
4.4.3 The Configuration Process ..... 92
4.5 Performance and Resource Utilisation Evaluation ..... 94
4.5.1 Resource Utilisation Evaluation ..... 94
4.5.2 Standard Configuration Operations Performance Evaluation ..... 96
4.5.3 Online Black-Box Bitstream Generation ..... 100
4.5.4 The Multiple-Clone Configuration Technique ..... 101
4.6 Chapter Conclusion ..... 102
**Chapter 5 : Reliability-Centric Internal Configuration Management ..... 104**
5.1 The Design of a Fault-Tolerant ICM ..... 105
5.1.1 Triple Modular Redundancy (TMR) ..... 105
5.1.2 Dual Modular Redundancy (DMR) ..... 106
5.1.3 Operation Monitor ..... 107
5.1.4 Resource Utilisation vs. Performance ..... 109
5.2 Soft-Error Handling Strategies ..... 110
5.2.1 Internal Readback Scrubbing ..... 110

5.2.2 External Configuration Memory Scrubbing ..... 115
5.2.3 Configuration Memory Scrubbing Evaluation ..... 118
5.3 Permanent-Fault Handling Strategies ..... 120
5.3.1 General Fault Mitigation Scheme ..... 120
5.3.2 Fast and Scalable BIST Diagnosis ..... 123
5.3.3 BIST Diagnosis Evaluation ..... 127
5.4 The Reliable Reconfigurable Real-Time Operating System ..... 128
5.4.1 R3TOS Architecture ..... 130
5.4.2 Online Routing ..... 131
5.4.3 HT Management ..... 133
5.5 Chapter Conclusion ..... 136
**Chapter 6 : An R3TOS-based Reliable and Secure Encryption Engine ..... 138**
6.1 Background on FPGA Security ..... 139
6.1.1 Basic Security Features in Commercial SRAM FPGAs ..... 139
6.1.2 Side Channel Attacks: Vulnerabilities and Countermeasures ..... 140
6.2 Overview of the Encryption Engine ..... 142
6.2.1 The Relocatable Cryptographic Core ..... 144
6.2.2 Online Placement of Heterogeneous Cores ..... 145
6.2.3 Configuration Management and Task Execution ..... 153
6.3 Proof-of-Concept Implementation ..... 157
6.3.1 Implementation of a Test Relocatable Cryptographic Core ..... 157
6.3.2 Implementing the Static Control Logic ..... 162
6.4 Experimental Results ..... 166
6.4.1 Task Allocation ..... 167
6.4.2 Configuration and Control of the Relocatable Cores ..... 168
6.4.3 Task Data Transfer ..... 170
6.4.4 Fault Detection and Recovery ..... 171
6.4.5 Task Execution Time Overhead ..... 171
6.5 Chapter Conclusion ..... 172
**Chapter 7 : A DPR-based Platform for Frequent Itemset Mining Acceleration ..... 174**
7.1 Background on Frequent Itemset Mining ..... 175
7.1.1 Background on FIM Algorithms ..... 176
7.1.2 FPGA Implementations of FIM Algorithms ..... 179

7.2 Overview of Proposed System ..... 182
7.2.1 Acceleration Task1: Item Support Counting ..... 183
7.2.2 Acceleration Task2: Item Sorting ..... 185
7.2.3 Acceleration Task3: Database Pruning ..... 187
7.2.4 Acceleration Task4: Sorting Database Transactions ..... 188
7.2.5 Acceleration Task5: Itemset Counting ..... 190
7.3 Implementation and Resource Utilisation ..... 194
7.4 Experimental Results ..... 199
7.4.1 Item Counting ..... 200
7.4.2 Sorting the Frequent Items ..... 201
7.4.3 Database Pruning ..... 201
7.4.4 Sorting Database Transactions ..... 202
7.4.5 Itemset Counting ..... 203
7.5 Chapter Conclusion ..... 205
**Chapter 8 : Conclusion and Future Work ..... 206**
8.1 Summary and Concluding Remarks ..... 207
8.2 Future Work ..... 211
**References ..... 215**

# List of Figures

Figure 2.1 Virtex4 device architecture [25] ..... 14
Figure 2.2 Virtex-4 CLB and slice architecture [25] ..... 15
Figure 2.3 Virtex-4 BRAM architecture [25] ..... 16
Figure 2.4 Routing lines and interconnect pattern in Virtex-4 [26] ..... 18
Figure 2.5 Global clock nets and BUFGs in central column [27] ..... 19
Figure 2.6 Simplified regional clock distribution [27] ..... 19
Figure 2.7 Simplified Xilinx design flow ..... 21
Figure 2.8 Partial reconfiguration in SRAM-FPGAs ..... 22
Figure 2.9 Simplified Xilinx DPR flow ..... 24
Figure 2.10 Reserving static routes in modular DPR ..... 25
Figure 2.11 Limitations of Xilinx DPR flow ..... 26
Figure 2.12 Simplified Altera DPR flow ..... 27
Figure 2.13 Feasible RM relocation ..... 30
Figure 2.14 Reserving static routes for RM relocation ..... 31
Figure 2.15 LUT-based BMs ..... 33
Figure 2.16 On-chip communication infrastructures for relocatable RMs ..... 33
Figure 3.1 DPR-based systolic array acceleration ..... 41
Figure 3.2 Enhanced software acceleration with DPR [71] ..... 41
Figure 3.3 2-D task allocation algorithms [85] ..... 45
Figure 3.4 RM pre-fetching ..... 50
Figure 3.5 DPR-based fault repair in a redundancy system ..... 57
Figure 3.6 Basic BIST circuit [125] ..... 59
Figure 3.7 Roving fault detection ..... 59
Figure 3.8 Circumventing damaged resources ..... 60
Figure 4.1 Building blocks of the ICM ..... 65
Figure 4.2 Building blocks of the ICAP controller ..... 68
Figure 4.3 HWICAP based configuration systems [88] ..... 70
Figure 4.4 The dual-port BRAM block ..... 73
Figure 4.5 Writing three identical consecutive frames with and without compression ..... 76
Figure 4.6 Main states in the data transfer phase ..... 78
Figure 4.7 Transfer phase for read/write operations ..... 80
Figure 4.8 Transfer phase for basic partial reconfiguration ..... 81
Figure 4.9 Frame addressing in Xilinx Virtex FPGAs ..... 82
Figure 4.10 Transfer phase when relocating to the bottom half of the FPGA ..... 84
Figure 4.11 Transfer phase for black-box configuration ..... 85
Figure 4.12 The external memory controller ..... 87
Figure 4.13 Possible applications of the multiple-clone configuration ..... 90
Figure 4.14 Multiple-clone configuration ..... 91
Figure 4.15 The clonable bit file ..... 92

Figure 4.16 Configuration using the clonable bit file ..... 93
Figure 5.1 TMR design for the ICM ..... 106
Figure 5.2 DMR design for the ICM ..... 107
Figure 5.3 CRC error detection in the ICM ..... 108
Figure 5.4 ECC logic block in a Virtex-4 FX12 FPGA ..... 111
Figure 5.5 Bit indexing in a configuration frame ..... 112
Figure 5.6 Configuration data mismatch between ICAP and ECC logic ..... 114
Figure 5.7 The ICM scrubbing read operation ..... 115
Figure 5.8 Online CRC for external scrubbing ..... 117
Figure 5.9 Scrubbing time overhead ..... 118
Figure 5.10 Combined external and readback scrubbing schemes ..... 119
Figure 5.11 Permanent-fault mitigation ..... 121
Figure 5.12 Fault diagnosis ..... 122
Figure 5.13 Tiling the relocatable BIST circuits ..... 124
Figure 5.14 ORA implemented with a 3-input LUT and a flip-flop [125] ..... 125
Figure 5.15 CUT, ORA and TPG arrangement in BIST circuits ..... 126
Figure 5.16 Configuration time in BIST diagnosis ..... 127
Figure 5.17 Storage memory required for BIST configurations ..... 128
Figure 5.18 R3TOS [17] ..... 129
Figure 5.19 Simplified R3TOS architecture ..... 130
Figure 5.20 ICAP-based data transfer [144] ..... 132
Figure 5.21 Fixed clock distribution [27] ..... 133
Figure 5.22 The relocatable module architecture ..... 133
Figure 5.23 HT execution management ..... 134
Figure 5.24 Relocatable module's FSM operation ..... 135
Figure 6.1 R3TOS cryptography server ..... 143
Figure 6.2 Generic architecture of the relocatable block cipher ..... 144
Figure 6.3 Mapping FPGA resources into a resource matrix ..... 147
Figure 6.4 Offset groups for relocatable bitstream consisting of CLB and BRAM resources in a Virtex-4 FX60 FPGA ..... 148
Figure 6.5 Core horizontal layout's compatibility with offset groups ..... 148
Figure 6.6 Placement scheme with task reuse support ..... 153
Figure 6.7 Secure configuration of relocatable cipher cores ..... 154
Figure 6.8 Simplified operation of the system ..... 155
Figure 6.9 Multiple-clone configuration of the same cipher core ..... 157
Figure 6.10 The PRESENT cipher block diagram [174] ..... 158
Figure 6.11 Data mapping in the cryptographic core ..... 159
Figure 6.12 Relocatable core's LUT semaphores ..... 159
Figure 6.13 Checksum LUT ..... 160
Figure 6.14 Resource layout of the relocatable cipher ..... 162
Figure 6.15 15 Input, output and CRC LUTs mapping in the 19$^{th}$ frame of the first CLB column ..... 162
Figure 6.16 Simplified diagram of system's components ..... 163
Figure 6.17 Floor-plan image of the control logic in the system ..... 165
Figure 6.18 Initialisation of the FPGA's resource matrix ..... 166
Figure 6.19 Task configuration ..... 168

Figure 6.20 Task removal time ..... 169
Figure 6.21 Maximum task wait time ..... 172
Figure 7.1 Creating the FP-tree ..... 178
Figure 7.2 Previously proposed 2-D systolic tree for FIM [63] ..... 181
Figure 7.3 Database format in memory ..... 183
Figure 7.4 1-D systolic array ..... 184
Figure 7.5 3-item 2-D systolic array ..... 191
Figure 7.6 Itemset counting using the proposed 2-D systolic array ..... 193
Figure 7.7 The Static implementation of the system ..... 196
Figure 7.8 The DPR-based implementation of the system ..... 197
Figure 7.9 Floorplan images of the two implementations in a Virtex-6 LX270 ..... 199
Figure 7.10 Database pruning time overhead ..... 202
Figure 7.11 Itemset support calculation time overhead for top-k items ..... 204
Figure 7.12 Support counts calculated in hardware ..... 204

# List of Tables

Table 2.1 Configuration ports [25] ..... 28
Table 3.1 Characteristics of hardware and software tasks [72] ..... 42
Table 3.2 Soft-error detection/correction capabilities in different FPGAs ..... 54
Table 4.1 Main configuration operations ..... 67
Table 4.2 Xilinx configuration command structure [25] ..... 71
Table 4.3 Writing command templates ..... 74
Table 4.4 Reading command templates ..... 75
Table 4.5 MFW command templates ..... 77
Table 4.6 ICM's resource utilisation in a Virtex-4 FPGA ..... 95
Table 4.7 Resource utilisation for different versions of the ICM ..... 95
Table 4.8 Resource utilisation comparison between proposed ICM and HWICAP based systems in a Virtex-4 FPGA ..... 96
Table 4.9 Frame read/write time overhead ..... 97
Table 4.10 Benchmark RMs ..... 97
Table 4.11 Configuration and relocation times of the ICM ..... 98
Table 4.12 Throughput comparison between the proposed ICM and other relocation systems ..... 99
Table 4.13 RM removal time using (SRAM controller) ..... 100
Table 4.14 Test relocatable cores ..... 101
Table 4.15 Configuration times of the test cores ..... 102
Table 5.1 Resource utilisation of different versions of the FT-ICM in a Virtex-4 FPGA ..... 109
Table 5.2 Area occupation and recovery time in a Virtex-4 FX60 FPGA ..... 109
Table 5.3 Errors in critical design components ..... 110
Table 5.4 ECC syndrome decoding ..... 111
Table 5.5 Truth table for the ORA's 3-input LUT ..... 125
Table 6.1 Resource utilisation of the PRESENT cipher in a Virtex-4 FPGA ..... 158
Table 6.2 Resource utilisation of the relocatable cryptographic core in a Virtex-4 FPGA ..... 160
Table 6.3 Task allocation time overhead ..... 167
Table 6.4 Task control time overhead breakdown ..... 169
Table 6.5 Task input data transfer time ..... 170
Table 6.6 Task output data transfer time ..... 170
Table 6.7 Fault detection and recovery time overhead ..... 171
Table 6.8 Test cipher core execution time ..... 171
Table 7.1 Example database ..... 176
Table 7.2 Support count for itemsets in the example database ..... 176
Table 7.3 Itemsets generated from the FP-tree ..... 179
Table 7.4 Summary of acceleration tasks in the proposed system ..... 182
Table 7.5 CLB resource utilisation of the 1-D array in a Virtex-6 LX270 FPGA ..... 194
Table 7.6 Maximum operating frequencies for the 1-D array ..... 195

Table 7.7 Resource utilisation of the 2-D array in a Virtex-6 LX270 FPGA ..... 195
Table 7.8 Static logic resource utilisation in a Virtex-6 LX270 ..... 198
Table 7.9 Comparison between the two system implementations ..... 198
Table 7.10 Benchmark databases [186] ..... 199
Table 7.11 Item counting time overhead ..... 200
Table 7.12 Item sorting time overhead ..... 201
Table 7.13 Time overhead for sorting database transactions ..... 203

# List of Algorithms

Algorithm 4.1 Calculating relocation row address from Y offsets ..... 83
Algorithm 5.1 Error index calculation when S [11] = 1 and S [10:0] ≠ 0..... 112
Algorithm 5.2 Error index calculation when S [11] = 1 and S [10:0] = (0 or $2^n$) ..... 113
Algorithm 6.1 Vertical scan of the resource matrix ..... 150
Algorithm 6.2 Resource matrix scan..... 151
Algorithm 6.3 Update resource matrix ..... 151
Algorithm 7.1 Item support counting..... 184
Algorithm 7.2 Item list sorting algorithm ..... 186
Algorithm 7.3 Initialising PEs with frequent items ..... 187
Algorithm 7.4 Assigning order numbers to items entries in the database..... 188
Algorithm 7.5 Sorting items in database transactions..... 189
Algorithm 7.6 Initialising the 2-D systolic array ..... 192
Algorithm 7.7 Calculating the support count of itemsets ..... 192
Algorithm 7.8 Shifting items in the same level out of the 2-D systolic array..... 193

# List of Acronyms and Abbreviations

| | |
|---|---|
| **ACS** | Adaptive Computing System |
| **AES** | Advanced Encryption Standard |
| **ASIC** | Application Specific Integrated Circuit |
| **BF** | Best-Fit |
| **BIST** | Built-In Self-Test |
| **BM** | Bus-Macro |
| **BRAM** | Block Random Access Memory |
| **CLB** | Configurable Logic Block |
| **CPU** | Central Processing Unit |
| **CRC** | Cyclic Redundancy Check |
| **CUT** | Circuit-Under-Test |
| **DB** | Dielectric Breakdown |
| **DCM** | Digital Clock Manager |
| **DDR** | Double Data Rate |
| **DM** | Deadline Monotonic |
| **DMA** | Direct Memory Access |
| **DMR** | Dual-Modular Redundancy |
| **DVF** | Device Vulnerability Factor |
| **DPR** | Dynamic Partial Reconfiguration |
| **DRC** | Design Rule Checking |
| **DSP** | Digital Signal Processing |
| **ECC** | Error Correction Code |
| **EDA** | Electronic Design Automation |
| **EDF** | Earliest Deadline First |
| **FAC** | Frame Address Calculator |
| **FAR** | Frame Address Register |
| **FCFS** | First-Come-First-Serve |
| **FDRI** | Frame Data Register-Input |
| **FDRO** | Frame Data Register-Output |
| **FF** | First-Fit |
| **FIM** | Frequent Itemset Mining |
| **FPGA** | Field-Programmable Gate Array |
| **FSL** | Fast Simplex Link |
| **FSM** | Finite State Machine |
| **GPP** | General Purpose Processor |
| **GPU** | Graphical Processing Unit |
| **GRM** | General Routing Matrix |
| **HCI** | Hot Carrier Injection |

| | |
|---|---|
| **HDL** | Hardware Description Language |
| **HLS** | High-Level Synthesis |
| **HPC** | High Performance Computing |
| **HPRC** | High Performance Reconfigurable Computing |
| **HT** | Hardware Task |
| **HWICAP** | Hardware Internal Configuration Access Port |
| **ICAP** | Internal Configuration Access Port |
| **ICM** | Internal Configuration Manager |
| **IDB** | Input Data Buffer |
| **IOB** | Input-Output Block |
| **IP** | Intellectual Property |
| **IPR** | Impossible Placement Region |
| **JPL** | Jet Propulsion Lab |
| **KDD** | Knowledge Discovery and Data Mining |
| **LUT** | Look-Up Table |
| **MBU** | Multiple Bit Upset |
| **MER** | Maximum Empty Rectangle |
| **MFW** | Multiple Frame Write |
| **MPMC** | Multi-Port Memory Controller |
| **MSF** | Masked SRAM object File |
| **MTTD** | Mean Time To Detect |
| **MTTM** | Mean Time To Manifest |
| **MTTR** | Mean Time To Repair |
| **NCD** | Native Circuit Description |
| **NGD** | Native Generic Database |
| **NoC** | Network-on-Chip |
| **NOP** | No-Operation |
| **NPI** | Native Peripheral Interconnect |
| **ODB** | Output Data Buffer |
| **OpenCL** | Open Computing Language |
| **ORA** | Output-Response-Analyser |
| **OS** | Operating System |
| **PAR** | Place and Route |
| **PE** | Processing Element |
| **PIM** | Personality Interface Module |
| **PIP** | Programmable Interconnection Point |
| **PMSF** | Partial-Masked SRAM object File |
| **PWM** | Pulse Width Modulation |
| **ROM** | Read-Only Memory |
| **ROS** | Reconfigurable Operating System |
| **R3TOS** | Reliable Reconfigurable Real-Time Operating System |
| **RM** | Reconfigurable Module |
| **RTL** | Register-Transfer Level |
| **RP** | Reconfigurable Partition |
| **RUT** | Region Under Test |
| **SB** | Switch Box |

| | |
|---|---|
| **SEM** | Soft Error Mitigation |
| **SER** | Soft Error Rate |
| **SET** | Single Event Transient |
| **SEU** | Single Event Upset |
| **SoC** | System-on-Chip |
| **SOF** | SRAM Object File |
| **SRAM** | Static Random Access Memory |
| **TB** | Tristate Buffer |
| **TPG** | Test-Pattern-Generator |
| **TMR** | Triple-Modular Redundancy |
| **UCF** | User Constraint File |
| **VLS** | Vertex List Set |
| **XST** | Xilinx Synthesis Technology |
| **ZBT** | Zero Bus Turnaround |

# Chapter 1 : Introduction

It is difficult to comprehend the evolution of the electronics industry and how technology has changed every aspect of our lives. Embedded computing devices are being deployed everywhere, from large machinery to small consumer products. The diverse market and high demand have been the driving force for the massive growth in the electronics industry, which currently ranks top in research and development spending among all other industries [1]. After the introduction of the first commercial silicon transistor in 1954 [2], and throughout the history of computing hardware, the transistor count in computing hardware has remained more or less on par with Moore's law. The continuous advances in manufacturing process technology have allowed for computing hardware such as processors to have better transistor density, higher performance and greater energy efficiency.

Processors are based on the stored-programme computing model whereby programme instructions for a given function are stored in memory. These instructions are executed sequentially in repetitive fetch-decode-execute cycles. Processors are associated with software; the ease and convenience of writing software programs has boosted the productivity of processor-based computing making it the most dominant trend in computing. In fact, the software industry today generates over $400 billion in revenue annually due to the widespread use of processor-powered devices [3].

Most processors are classified as General Purpose Processors (GPPs). GPPs are designed for a wide range of applications. They are very flexible, especially when running an Operating System (OS) to manage all the hardware resources and peripherals. Due to the sequential nature of programme execution in GPP-based systems, their performance is highly related to their operational clock frequency. In

early processors, manufacturers relied on the advancements in process technology to increase the operational clock frequency within the acceptable power envelope. This trend in clock frequency scaling continued until the mid-2000s when the transistor shrink could no longer compensate for thermal and power dissipation. To continue scaling the performance of processors, manufacturers slowly adopted multicore architectures that integrate multiple cores in a single processor so that multiple instructions can be executed simultaneously. Multicore architectures have become the norm today and can be seen in a wide range of processors from large high-performance processors to small low-energy application processors. The performance of multicore processors does not necessarily scale with the number of cores in the processor. Performance is highly related to the software implementation and its ability to parallelise the computing tasks. The performance gain of multicore processors is governed by Amdahl's law, which states that the performance gain is limited by the portion of software that can be parallelised to run on multiple cores simultaneously [4]. Currently, parallel programming is a hot topic of research with many emerging programming models and tools [5]. However, performance gain is not always possible with parallel programming as it depends highly on the application. Extracting parallelism from algorithms is not always a trivial process. In addition, parallel programming is not mature enough to be deployed on a wider scale in an industry dominated by 'serially-orientated' software and hardware [6].

GPPs are intended for general use; they might not provide sufficient power for applications requiring high performance and throughput. Such applications are usually handled by specialised Application Specific Integrated Circuits (ASICs) such as Digital Signal Processing (DSP) accelerators and Graphical Processing Units (GPUs). Depending on the application, modern processors usually work alongside one or several ASICs that are optimised for specific computation tasks. These ASICs can be independent chips connected externally to the processor or Intellectual Property (IP) blocks integrated with the processor in a single System-on-Chip (SoC). While ASICs greatly enhance performance by customising the hardware to the needs of specific applications, they compromise flexibility. Once fabricated, ASICs can

only be used for certain types of task and this has limited their deployment to a limited number of applications.

In general, software provides the best flexibility, whereas custom hardware provides the best performance. Reconfigurable computing has become one of the major computing trends to bridge the gap between software and custom hardware in terms of flexibility and performance. Reconfigurable computing is based on reconfigurable hardware such as Field-Programmable Gate Arrays (FPGAs). As the name implies, FPGAs are built out of arrays of reconfigurable 'logic blocks', which can be programmed after manufacturing. Almost any digital hardware circuit can be implemented in FPGAs with a performance comparable to that of fully-custom ASICs. While early FPGAs were small devices used to implement simple 'glue logic', modern FPGAs have taken full advantage of the advances in the manufacturing process to become large and highly dense devices containing many specialised components and capable of hosting very complex systems in a single chip. In fact, the transistor count can exceed 20 billion transistors in a modern FPGA [7]. This huge amount of reconfigurable blocks is enough to emulate 10 ARM Cortex-A9 CPUs in a single chip.

The FPGA design flow is also rapidly evolving to allow for better productivity and shorter development time. Traditionally, FPGA design starts with a Register-Transfer Level (RTL) description typically written in a Hardware Description Language (HDL). The FPGA industry today is shifting towards a higher level of abstraction in design. C-to-silicon design tools allow for designing FPGA systems using software programming languages without the need for creating the RTL description manually. The main FPGA vendors are starting to push these tools, which are gaining a lot of popularity commercially and academically. Xilinx, the primary FPGA vendor, was first to introduce the High-Level Synthesis (HLS) tool, which supports rapid FPGA design using a broad range of programming languages [8]. Altera, Xilinx's main competitor, soon followed, introducing an FPGA compiler for the popular Open Computing Language (OpenCL). This compiler automatically extracts and translates OpenCL kernels into deeply pipelined hardware accelerators [9]. With the variety of design tools available for FPGAs, the FPGA IP market is rapidly expanding. Both

Xilinx and Altera are making their design tools more IP user-friendly. In addition, big Electronic Design Automation (EDA) companies such as Mentor Graphics and Synopsys have introduced vender-independent IP platforms, allowing access to libraries from several third-party IP vendors ([10] and [11]).

FPGAs are all about flexibility when compared to ASICs. The advantage of FPGAs is not just limited to the flexibility of implementing different designs and the ability to perform offline design modifications. The Dynamic Partial Reconfiguration (DPR) feature in high-end SRAM FPGAs allows for certain blocks within the FPGA to be modified at run-time while the FPGA is operating. DPR can enhance the efficiency of FPGAs by allowing different functions to share the same hardware and consequently reduce the overall resource utilisation. DPR can also enhance performance by time-multiplexing larger functions that share the same hardware resources. In fact, DPR takes the flexibility of FPGAs to another level and presents some exciting opportunities to implement new computing architectures that effectively exploit the FPGA's resources. The idea of a Reconfigurable Operating System (ROS) has existed for a long time [12]. An ROS would have all the flexibility and productivity benefits of a normal OS; however, tasks are executed using reconfigurable hardware rather than software, allowing for much higher performance. The lack of a generic hardware platform that allows for continuous run-time modifications has prevented this idea from materialising into practical systems. With today's advancements in FPGA technology and DPR techniques, this idea is more feasible than ever.

FPGAs allow for in-field repairs, modifications and upgrades, making them very attractive for space and military electronics. However, space and military electronics operate in harsh conditions, which can provoke faults in FPGAs. FPGAs are particularly sensitive to high levels of radiation. The reconfigurability of FPGAs compensates for this fact as continuous repairs and workarounds are possible at virtually no cost. Moreover, DPR can be performed from within the FPGA without the aid of any external control circuitry. This feature allows for implementing self-healing evolvable systems that adapt not only to temporarily faults but also to permanent faults caused by device aging.

Unfortunately, the current state of DPR design flows and configuration techniques does not unleash the full potential of this feature. Particularly, the lack of efficient configuration management systems limits the adoption of this feature in many applications that would naturally benefit from DPR. The main aim of this thesis is to develop an Internal Configuration Manager (ICM) that would enable efficient deployment of DPR for high performance and reliability. This thesis looks into a wide range of DPR issues such as: configuration speed, configuration reliability, DPR design floor planning, reconfigurable module allocation, fault detection and fault recovery. Moreover, the thesis demonstrates two DPR case study applications; the first is a reliable encryption engine based on an ROS model for FPGAs and the second is a high-performance acceleration engine for frequent itemset mining.

## 1.1 Thesis Objectives

The main objective of the thesis is to propose and develop ways that would unlock the full potential of DPR and to come up with generic reconfiguration platforms and configuration techniques that would enable efficient use of the reconfigurable hardware for several applications. In more detail, the objective of the thesis can be divided into the following two areas of interest:

- **High Performance**

DPR can theoretically enhance performance by efficient exploitation of the reconfigurable resources at run-time. In general, performance gain is achieved by time-multiplexing several reconfigurable accelerators that share the same reconfigurable resources. Three questions need to be answered to come up with the optimal performance gain using DPR:

1) ***Can we reduce reconfiguration overhead through efficient management of reconfiguration tasks?***

   Reconfiguration is a sequential process with a throughput limited by the rated maximum clock frequency. In many cases, reconfiguration can be a performance bottleneck preventing any feasible performance gain out of DPR. This thesis

aims at developing new configuration techniques that allow for higher configuration throughputs. Ultimately, a generic internal configuration controller should be able to efficiently handle all the configuration operations in any system.

2) ***Can a generic DPR-based computational platform be used for software acceleration?***

   This thesis aims to explore how efficient configuration management and high-speed reconfiguration would affect the overall performance of the system. This thesis also aims to develop a generic platform for high-performance acceleration through DPR. This platform should be applicable to a wide range of computationally intensive algorithms.

3) ***Can DPR provide an actual gain in performance in a real-world application?***

   This thesis demonstrates a real-world application in which DPR is applied to gain a performance advantage over static implementations. Customised reconfigurable accelerators are developed for the selected applications and deployed in the generic acceleration platform.

- **Reliability**

DPR can be the basis for implementing self-healing reliable systems. Using DPR for reliability requires addressing several aspects in the system. This thesis aims to develop a reliability-centric configuration management system that addresses the following open problems:

1) ***Can DPR be used for effective transient fault detection and correction?***

   The thesis aims to develop a comprehensive transient faults handling scheme that takes into account both fault detection and correction in the static and reconfigurable parts of the system. The scheme should efficiently utilise the configuration port of the target device for continuous repairs by means of conventional memory scrubbing techniques as well as DPR.

2) ***Can DPR be used for effective permanent fault detection and correction?***
   The fault handling scheme should consider permanent damage in the target device's reconfigurable fabric and be able to work around the affected resources. This fault handling scheme should extend the life-time of the device and make it suitable for long missions in harsh environments. The thesis aims to develop a scalable self-test mechanism to test the reconfigurable resources at run-time by loading specialised testing circuits.

3) ***Can a single system be used for both transient and permanent fault mitigation?***
   Designing DPR applications is not a trivial process, especially when fault-tolerance is a main objective in the design. This thesis aims to pave the way for a Reliable Reconfigurable Real-Time Operating System (R3TOS) that can offload computation tasks to specialised hardware modules. This ROS should naturally handle both transient and permanent faults and guarantee reliable execution of the computation tasks. The thesis also aims to demonstrate the main functionalities of R3TOS with a case study application that requires reliability and high-performance.

## 1.2 Novelty and Contribution

First of all, this thesis presents the design and architecture of a novel ICM for Xilinx FPGAs that supports a wide range of configuration operations [13]. The ICM is highly portable and is optimised for efficiency and high throughput. In addition, the ICM can act as a fast bitstream manipulation filter based on a novel offset-based relocation model. Moreover, the novel multiple-clone configuration technique is fully integrated into the ICM, allowing for high throughputs that can be multiple times greater than the maximum theoretical throughput rated for the internal configuration port for Xilinx FPGAs [14].

A novel fault-tolerant version of the ICM is also presented in the thesis [15]. The ICM can detect and recover from faults in its logic. This fault-tolerant ICM is the core component in R3TOS ([16] and [17]).

In addition, this thesis presents a fault-handling scheme that addresses both transient and permanent faults in an ROS-like system. The scheme combines several fault mitigation techniques: memory scrubbing, modular redundancy and module relocation. Module relocation is heavily utilized to freely move computational modules around permanently damaged resources. Permanent fault detection and isolation is achieved by a novel Built-In Self-Test diagnosis scheme, which deploys the multiple-clone configuration technique to become considerably faster than the available FPGA online-testing techniques [18].

The thesis also presents a practical placement algorithm for relocatable modules based on an efficient online vertical scanning of the FPGA resources. The algorithm's main contribution is the support of heterogeneous module relocation as well as efficient module reuse.

Lastly, two case study applications are demonstrated in this thesis. The first is a flexible encryption engine implemented over R3TOS to provide a secure and reliable system for executing encryption tasks. The potential capabilities of the system are demonstrated with a test relocatable hardware cipher, which can be allocated in several locations in the FPGA to serve different concurrent encryption tasks. The second application is a novel DPR implementation of a frequent itemset counting system, which deploys efficient management of acceleration tasks to speed-up the itemset counting process. Acceleration tasks are performed using customised systolic array accelerators which are managed internally using the proposed ICM.

The work presented in this thesis is a part of the R3TOS project carried out by the System Level Integration Group (SLIG) at the University of Edinburgh. It is important to acknowledge the contributions of the other member in the group in the R3TOS project. Dr. Xabier Iturbe developed the R3TOS kernel. More specifically, he developed the scheduling and allocation algorithms. Dr. Chuan Hong coded these algorithms and implemented dedicated hardware scheduler and allocator. The R3TOS scheduling and allocation algorithms are not used in this thesis. However,

Dr. Xabier Iturbe, performed reverse engineering experiments to extract the functionality of some configuration bits in the Virtex-4 FPGA. The results of these experiments are used in Chapter 6. In addition, he developed the main mechanism for online clock routing as well as one of the techniques for controlling the relocatable cores. This technique is based on the LUT and BRAM semaphores and is presented in Chapter 5. Finally, Dr. Hana Hussain has kindly provided the HDL code for the K-means core used in the analysis of Chapter 4.

## 1.3 Thesis Outline

This thesis is composed of eight chapters. The remainder of this thesis is summarised as follows:

**Chapter 2: Introduction to FPGAs and Dynamic Partial Reconfiguration**
This chapter introduces the basics of FPGAs in terms of: architecture, reconfigurable resources and design flow. The industry's DPR design flow is also introduced with an overview of its limitations. This chapter also reviews the relevant research work that aims to overcome the limitations of the basic DPR flow and allow for more advanced partially reconfigurable systems.

**Chapter 3: Dynamic Partial Reconfiguration for High-Performance and Reliability**
This chapter is a literature review of the main trends in DPR deployment for high performance and reliability. The chapter addresses the different techniques for enhancing performance using DPR as well as techniques to speed up the reconfiguration process. This chapter also discuss the concept of an ROS and its implementation issues on FPGAs. Reliability of FPGAs and fault mitigation techniques are addressed in this chapter with special emphasis on DPR-based fault mitigation for transient and permanent faults.

**Chapter 4: A High-Performance Internal Configuration Manager**

This chapter presents the design and architecture of the first all-in-one ICM, which independently handles the configuration protocols and bitstream manipulation for module relocation. This chapter explains in detail the configuration process through the Internal Configuration Access Port (ICAP) in Xilinx FPGAs and how the configuration operations can be managed efficiently to allow for the maximum throughput.

**Chapter 5: Reliability-Centric Internal Configuration Management**

This chapter demonstrates how internal configuration can be steered towards reliability and fault-tolerance. Different fault detection and recovery methods through the ICAP are explained in detail. In addition, different design hardening techniques for the ICM are presented and evaluated. This chapter draws a comprehensive fault-handling scheme and an ROS configuration management system, which led to the development of the R3TOS.

**Chapter 6: An R3TOS-based Reliable and Secure Encryption Engine**

This chapter presents a practical case study application of R3TOS. The case study demonstrates how continuous encryption tasks can be executed using relocatable cipher blocks. This chapter presents some practical solutions for module relocation, including: on-chip communication, remote task redundancy voting, secure configuration and task allocation. The performance of the implemented system is evaluated against software when using a test encryption algorithm.

**Chapter 7: A DPR-based Platform for Frequent Itemset Mining Acceleration**

This chapter presents a DPR-based platform for accelerating the popular FP-growth algorithm, which is widely used for frequent itemset mining. In this case study, the FP-growth algorithm is broken into several acceleratable stages. The proposed platform manages the execution of several acceleration tasks using relocatable systolic array accelerators. The overall performance of the implemented system is evaluated against static implantations of the algorithm.

**Chapter 8: Conclusion and Future Work**

This chapter draws conclusions from the research presented in the thesis and points towards the remaining open problems and future work.

# Chapter 2 : Introduction to FPGAs and Dynamic Partial Reconfiguration

FPGAs are reconfigurable logic devices that can repeatedly be reconfigured (reprogrammed) to alter or change their functionality. While early FPGAs were basically built out of small arrays of reconfigurable blocks used to implement simple glue-logic circuits, modern FPGAs have evolved dramatically over the past two decades to become complex devices containing several types of reconfigurable resource and several specialised components that can be used to implement specialised SoCs very quickly and at a very low cost ([19] and [20]). Modern FPGA technology is attracting the attention of engineers to explore different applications that would benefit from the advantages offered by FPGAs over ASICs. The key advantage of FPGAs over ASICs is flexibility. Indeed, almost any digital circuit can be implemented using pre-fabricated FPGAs allowing for fast and low-cost development and short time-to-market. In addition the ability to reconfigure FPGAs means that FPGA-based systems can be upgraded on-field, which protects such systems from obsolescence and allows for adapting to emerging standards. Furthermore, recent high-end FPGAs take the flexibility of the device to another level by allowing for run-time modifications to the system implemented on the FPGA fabric using DPR. DPR allows for sub-blocks in the system to be modified or changed without disturbing the operation of the other blocks. The flexibility brought by DPR can be harnessed to improve several design aspects such as performance [21], functional density [22] and power utilization [23].

The rich features offered by modern FPGAs have contributed to the significant growth in the FPGA market, which has been dominated by two companies, namely, Xilinx and Altera. According to [24], the 2013 FPGA market was worth $4.5 billion compared to $2 billion in 2001. During this period, Xilinx has maintained a steady

lead in the market with around 45%-50% of the total market share compared to a 40%-45% market share for Altera. Together these two companies account for around 90% of the FPGA market.

Both Xilinx and Altera focus their attention on SRAM-based FPGA technology, which is the most common type of FPGA technology and the one that currently supports DPR. This chapter prepares readers who are not so familiar with SRAM-based FPGA technology to better understand the work presented in this thesis. With focus on the Xilinx Virtex FPGA family, this chapter presents an overview of the FPGA architecture and DPR design flows as found in the literature.

## 2.1 Xilinx FPGAs and Design Flow

The Xilinx Virtex FPGA family is the high-end FPGA family offered by Xilinx. This family of FPGAs has evolved since the introduction of the first Virtex FPGA in 1998. After the successful launch of the first Virtex FPGA, Xilinx has followed up with the Virtex-2, Virtex-2 pro, Virtex-4, Virtex-5, Virtex-6 and recently the Virtex-7. While earlier iterations of the Virtex FPGAs have followed an incremental path of evolution in terms of fabrication process technology and number of reconfigurable blocks in the device, the Virtex-4 FPGA marked a milestone in the Virtex family evolution and a major architectural change to the former iterations of the family. The renowned Virtex-4 architecture became the standard for the following iterations of the Virtex family, which focused on increasing the device density and integrating more specialised components while keeping the general resource layout and configuration architecture.

The Virtex-4 architecture divides the chip into several ‘clock regions’ (see Figure 2.1). Each clock region contains tiles of reconfigurable resources. These reconfigurable resources are organised in columns within each clock region in a similar arrangement. A column can contain a single type of reconfigurable resource and can be configured using a number of ‘configuration frames’. The size of the configuration frame is fixed for all columns and is equal to 1312 bits arranged as 41

words each of 32-bit. In more recent Virtex FPGAs, the size of the configuration frame is larger to accommodate for the extra resources in each column.

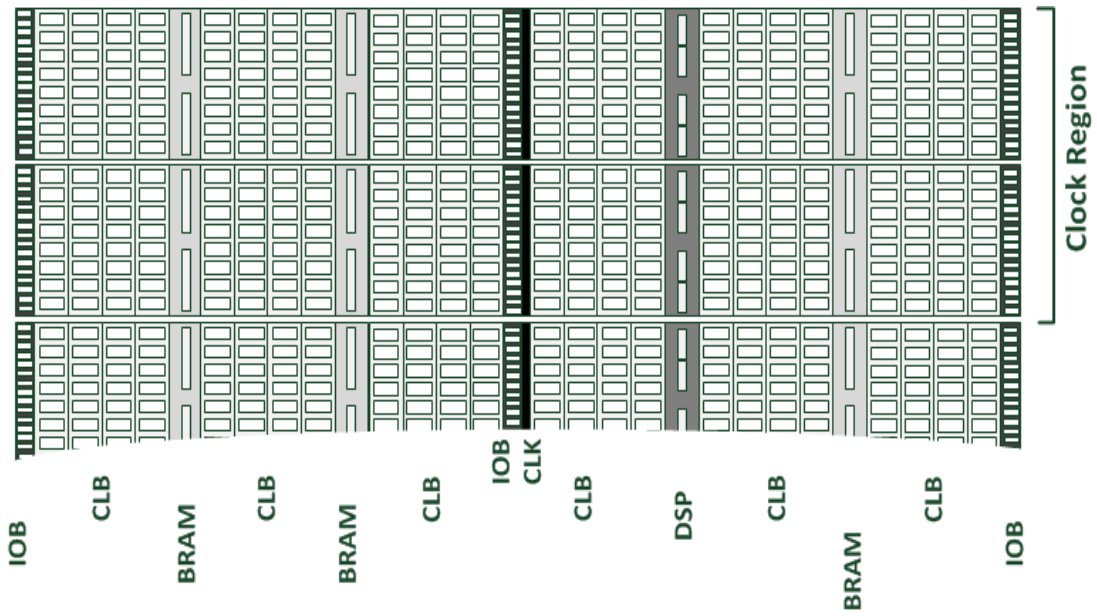



**Figure 2.1 Virtex4 device architecture [25]**

There are several standard types of reconfigurable resource, as seen in Figure 2.1. The main reconfigurable resources are: Configurable Logic Blocks (CLBs), Block RAM (BRAMs), Digital Signal Processing (DSP) Blocks, Input/Output Blocks (IOBs) and the clock management resources denoted by CLK. In addition to the reconfigurable resources, Xilinx FPGAs contains some hard-wired resources that have fixed locations on the chip and can be integrated with circuits implemented on the reconfigurable logic. These are referred to as primitives and include components such as processors, configuration ports and clock buffers.

### 2.1.1 Overview of Xilinx Reconfigurable Resources

This section describes the most relevant types of Xilinx reconfigurable resources, namely, the CLBs and the BRAM.

### The Configurable Logic Blocks

CLBs are the main type of reconfigurable logic resource in Xilinx FPGAs. Most of the Virtex FPGA fabric is composed of CLB columns. In Virtex-4 FPGAs, a CLB column consists of 16 vertically aligned CLBs. A CLB consists mainly of Look-Up Tables (LUTs), flip-flops and specialised carry-chains for direct connections with the top and bottom CLBs in the column. The LUT is the core element in a CLB. LUTs are memory components that can be initialised with the truth table of any function of its input connection. Virtex-4 FPGAs contain 4-input LUTs, which means it can be programmed to compute any logic function with up to four inputs. By connecting several of these LUTs, more complex functions or functions with more inputs can be implemented.

Each CLB is divided into four slices; two of these slices are of type SliceM and the other two are of type SliceL. Each slice contains two LUTs, two flip-flops and two carry-chains (see Figure 2.2). SliceL can only be used to implement 'Logic' functions. SliceM LUTs can be used to implement 'Memory' components such as shift registers and distributed RAM in addition to the basic logic functions.

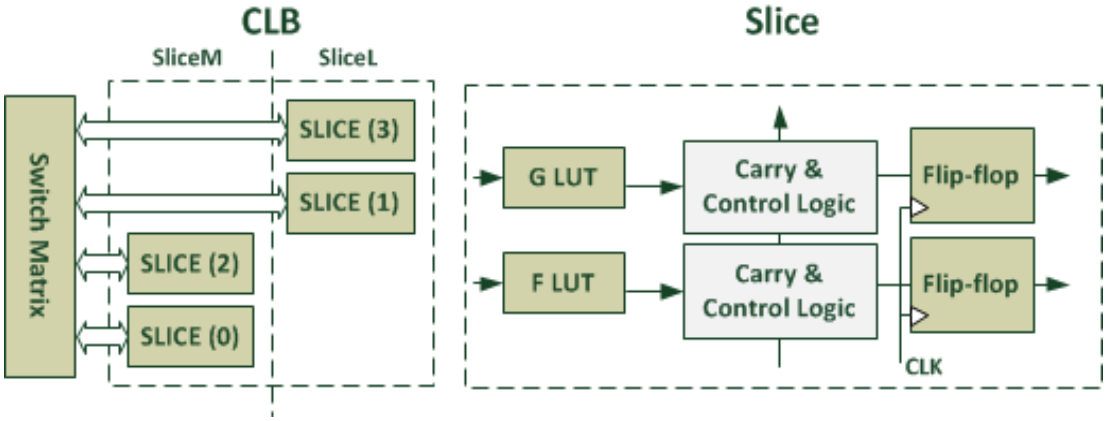



**Figure 2.2 Virtex-4 CLB and slice architecture [25]**

### The Block Random Access Memory

BRAMs are on-chip memory components organised in dedicated columns in the FPGA. Each column contains four BRAMs, and each can store 16Kb of data with an additional 2Kb of parity bits. Each BRAM can be configured as a single-port or dual-port memory and can also be configured with any memory location size from 16K x 1 to 512 x 32. Several BRAMs with the same configuration can be connected together to realise larger memory blocks. BRAMs content can be initialised in the HDL file, thus giving the option of implementing on-chip Read-Only Memory (ROM).

In addition to the core BRAM resources, a BRAM column also contains dedicated First-In First-Out logic, which enables the implementation of synchronous or asynchronous FIFOs (see Figure 2.3). This gives designers the option of implementing larger FIFOs without utilising any of the CLB resources.

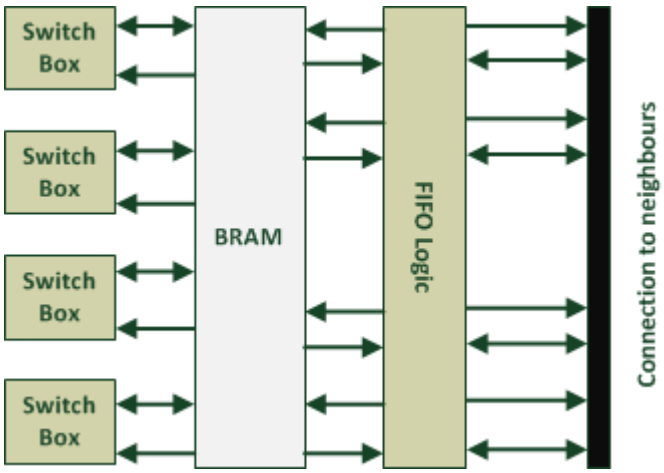



**Figure 2.3 Virtex-4 BRAM architecture [25]**

### 2.1.2 Basic Routing and Clocking Structure

Virtex FPGAs have a segmented and hierarchical routing structure. Most of the FPGA's configuration data is related to the routing of the internal logic. The fixed wiring in the FPGA can be divided into two categories: the logic routing lines and the clock nets. The logic routing lines connect the internal logic's signals, whereas the clock nets connect a clock signal to the resources of the FPGA.

**The Routing Structure**

Routing lines are configured by manipulating a routing structure called the General Routing Matrix (GRM). Resources in the FPGA are connected to the GRM via reconfigurable Switch Boxes (SBs). Routing lines in the GRM are divided into two types: Global lines and Local lines (see Figure 2.4a). Global lines can be one of two types of line: Long lines or Hex lines. Long lines connect SBs either vertically from the top to the bottom of the device or horizontally across the entire width of the device. Hex lines connect an SB to two neighbouring SBs located three and six positions away from this SB, either vertically or horizontally.

On the other hand, Local lines can be one of two types: Double lines or Direct lines. Double lines connect an SB to the first and second neighbouring SBs, either vertically or horizontally. Direct lines connect an SB to the first neighbouring SB, either vertically, horizontally or diagonally.

Activating a connection between a routing line and an SB is performed via programming a Programmable Interconnection Point (PIP) during the configuration of the device. An SB has several PIPs, one for each routing line connected to the SB. The PIP is basically a transistor switch that can be either enabled or disabled by a particular bit in the configuration file. To establish a particular connection in the SB, two PIPs must be enabled, one for the input and the other for the output. A connection between two SBs is referred to as a 'hop'. Multiple hops may be required to connect two SBs, depending on their location. Figure 2.4b shows the number of

hops required for connecting a particular SB to a central SB without considering the global lines.

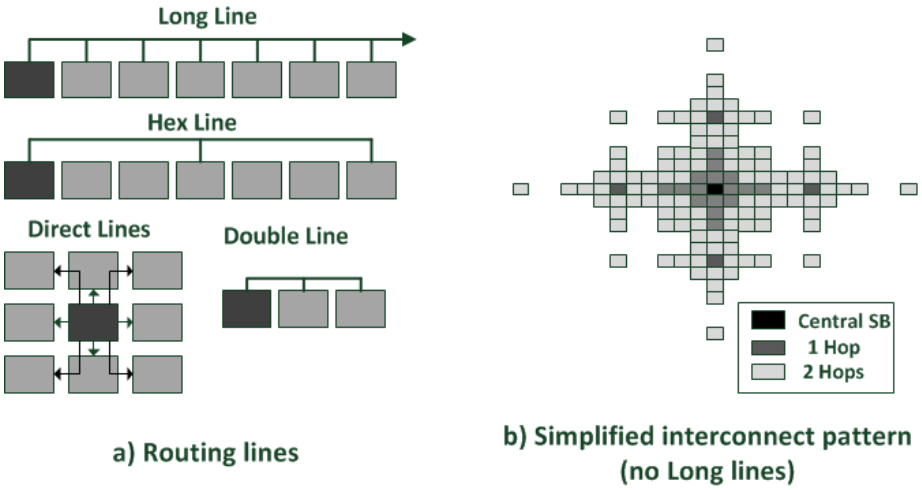



**Figure 2.4 Routing lines and interconnect pattern in Virtex-4 [26]**

### The Clock Tree

The clock routing in Virtex FPGAs is independent of the logic routing. In FPGAs, the clock tree is a fixed structure of nets and clocking resources that distribute the clock to the synchronous resources across the device. The clock resources are divided into global clocking resources, which drive the clock into dedicated global nets, and regional clocking resources, which drive the clock into dedicated regional nets within each clock region in the device [27]. The global clocking resources are typically located in the central columns of the device (see Figure 2.5a) where global clock buffers denoted by BUFGs are used to drive the clock to the global nets. An external clock source can be directly connected to a BUFG or can be connected first to a Digital Clock Manager (DCM), which can be used to adjust the frequency of the clock source (see Figure 2.5b).

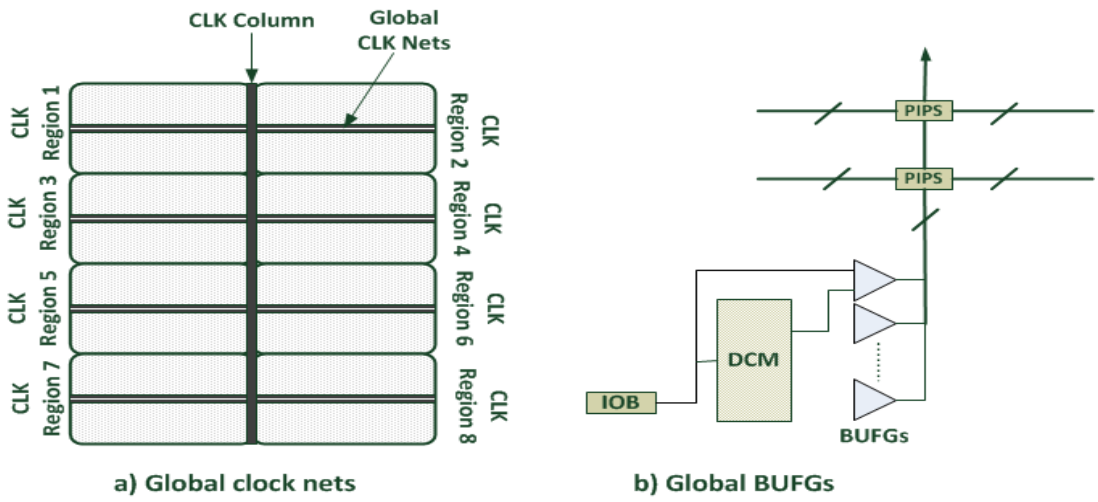



**Figure 2.5 Global clock nets and BUFGs in central column [27]**

Global clock nets can connect the clock directly to the resources of the FPGA or can connect the clock first to a regional clock buffer denoted by BUFR. Each clock region in the device contains two BUFRs; each one has a dedicated regional clock net (see Figure 2.6).

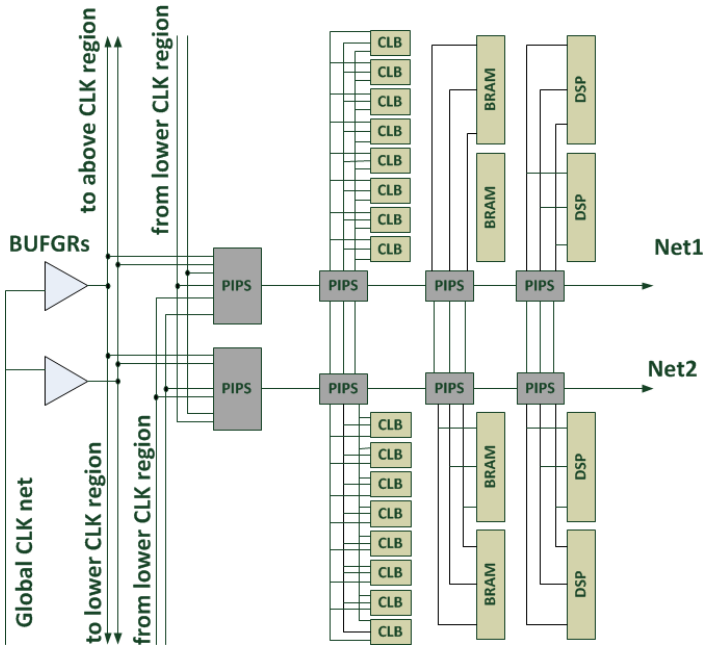



**Figure 2.6 Simplified regional clock distribution [27]**

In general, resources can be connected to any of the global or regional clock nets by programming the PIPs required to set the desired clock connections. If a regional net is desired to clock the resources of a particular implementation on the FPGA, the following sequence of resource is a possible path for the clock from the external source:

IOB → DCM → BUFG → PIPs → BUFR → PIPs → Resources

### 2.1.3 Basic Design Flow

In SRAM-FPGAs, the SRAM cells that hold the configuration of the device are referred to as the 'configuration memory'. Because SRAM is volatile, the configurations file (a.k.a. the bitstream) is usually stored in an external non-volatile memory module and is loaded into the FPGA's configuration memory after power-up of the device. Typically, FPGA designs start with HDL files written by the designer to describe the functionality of the logic to be implemented on the FPGA. After going through a number of design stages supported by the FPGA's vendor design tools, a bitstream is generated and can be loaded into the FPGA's configuration memory through one of the configuration ports of the device. In the Xilinx design flow, there are three main stages required to generate the bitstream:

**Design Synthesis:** In this stage, the HDL files described by the designers are converted into one or several netlists using the Xilinx Synthesis Technology (XST). A netlist file is denoted as the NGC and basically contains a generic hardware description of the implemented design (i.e. adders, multipliers, logic gates, etc.).

**Design Implementation:** This stage is composed of three design processes: the Translate, MAP and Place and Route (PAR). The Translate process merges all the NGC files into a single Native Generic Database (NGD). The NGD file is generated by the NGDBuild tool and contains a lower-level description of the hardware resources required in the target device to implement the design. The designer may direct the NGDBuild tool with specific constraints; these constraints may be the

exact locations of some of the required logic resources in the FPGA die. All of the design constrains are specified in the User Constraint File (UCF) before the translate process.

The second process in the design implementation stage is the MAP process, which physically maps all the logic defined in the NGD file to the FPGA resources such as CLBs and IOs. The MAP process generates the Native Circuit Description (NCD) file, which physically maps the design to the components of the FPGA.

The final process in the design implementation is the PAR, which takes the NCD file to generate another NCD file containing the final placed and routed design.

**Bitstream Generation**: After the final NCD file is created in the design implementation stage, the bitstream can be generated using the BITGEN tool, which generates a binary file denoted as the BIT file that represents the device configuration for the desired design. Figure 2.7 summarises the main design stages required to generate the BIT file for a particular design.

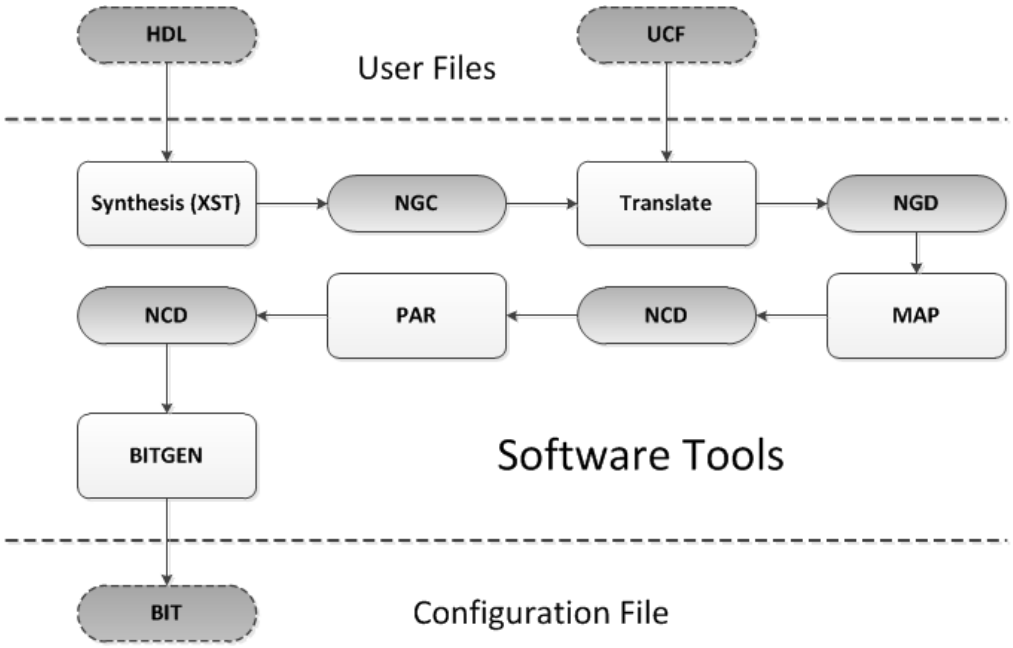



**Figure 2.7 Simplified Xilinx design flow**

## 2.2 Dynamic Partial Reconfiguration

DPR is an exclusive feature to SRAM-FPGAs where parts of the configuration memory are modified at run-time to alter the functionality of some parts of the implemented system. While a typical implementation on the FPGA has a single full bitstream loaded to the FPGA's configuration memory after power-up of the device, any implementation deploying DPR has a full bitstream as well as several partial bitstreams that correspond to the different configurations of the dynamically reconfigurable parts in the system (see Figure 2.8).

Since the introduction of DPR in some of the Xilinx devices of the mid-90s, the technology and software tools that support this feature have evolved dramatically. While this technology was limited to high-end Xilinx FPGAs a decade ago, most of the recent FPGAs introduced by Xilinx and Altera support DPR, making the technology widely available and a key feature of SRAM-FPGAs.

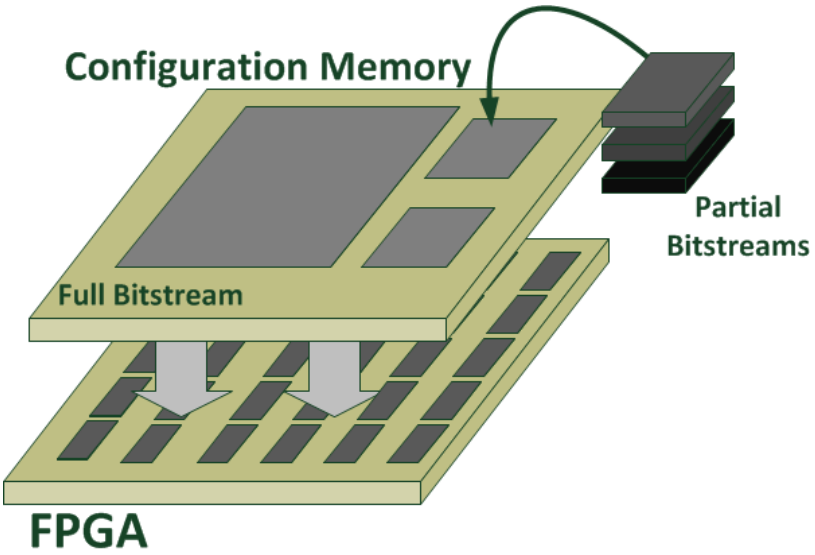



**Figure 2.8 Partial reconfiguration in SRAM-FPGAs**

### 2.2.1 Xilinx DPR Flow

In order to implement a partially reconfigurable design in a Xilinx FPGA, a design can follow the Xilinx DPR flow, which is supported by the PlanAhead software tool (see Figure 2.9) [28]. The DPR flow separates the design into two parts: the static logic, which does not change during run-time, and several Reconfigurable Modules (RMs), which are swapped in and out of the FPGA at run-time. Typically, the design flow starts with a top HDL file containing a hierarchical description of the entities in the design. The design may contain one or several reconfigurable entities that are reconfigured with RMs at run-time. Each RM in the design is described with a separate HDL file and is synthesised separately from the top HDL file to generate separate NGC files, one for each RM in addition to the top NGC file. Before the design implementation stage, the design is floor-planned using the PlanAhead tool. In floor-planning, each reconfigurable entity in the design is placed in a distinct reconfigurable region in the chip. The reconfigurable regions are often referred to as Reconfigurable Partitions (RPs). Each RP in the design is assigned with the desired RMs, if the design passes the Design Rule Checking (DRC), placement constraints can be created for the selected PRs.

In a DPR design, the implementation stage is repeated several times. Each implementation is referred to as a 'run'. Each run contains a different set of RMs assigned to the RPs in the design. The design with a particular set of RMs is implemented in the first run to create several NCD files, one for the static logic in the design and one for each RM in the design. The NCD file of the static logic from the first run is then reused for the remaining runs to ensure that no routing conflicts occur between the different implementations when generating the NCD files for the remaining RMs in the design.

In the bitstream generation stage, the static NCD file with the RM NCD files is used to generate full bitstreams, one for each run in the design, and partial bitstreams, one for each RM in the design. Depending on the nature of the design, the designer may select the type of the partial bitstream, which can be either a modular partial

bitstream or a difference-based partial bitstream. Modular partial bitstreams contain the full configuration of the RP area, whereas difference-based partial bitstreams contain individual configuration frames that perform minor changes in the RP area. Difference-based partial reconfiguration is suitable for designs with very similar RMs that differ only in terms of the content of some of its memory components (e.g. LUT equations, BRAM initialisation, etc.).

It is also noted that, in the case of modular partial reconfiguration, each RP can be assigned with an optional 'black-box', which is an empty module configured when an RP is not used by any RM to reduce the static power dissipation.

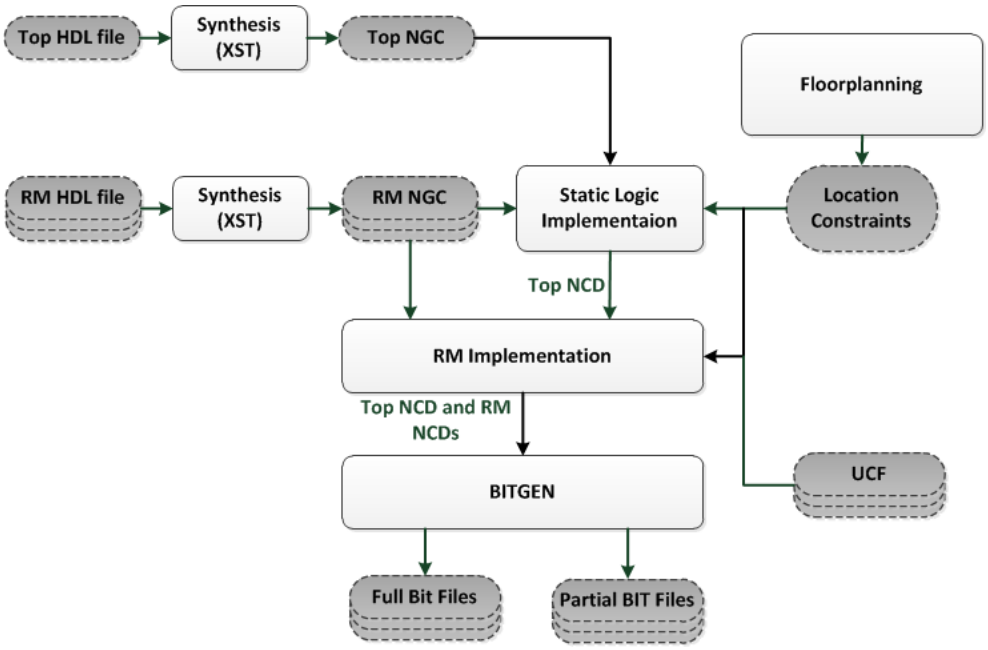



**Figure 2.9 Simplified Xilinx DPR flow**

### Limitations of Xilinx DPR Flow

Unfortunately, Xilinx DPR flow and software tools only support some of the capabilities that can be exploited in partially reconfigurable FPGAs. The main limitation of the Xilinx DPR flow is that designers do not have control over the static routes in the design. There are two types of route in a DPR design: the static route

which connects the different static components together; and the RM route which connects the local components inside each RM. Generally, the routing in Xilinx DPR flow is bound by three rules:

1) The static routes can pass through an RP. However, these routes needs to be reserved in each RM assigned to the RP. When an RM is configured the static routes are overwritten without disturbing the operation of the system (see Figure 2.10).
2) The local routes of any RM are confined within the area specified for the RP. This results in an average packing efficiency of around 80% for the PAR process in the Xilinx tools [29]. This means that the RP area must be at least 20% larger than that needed for the largest RM assigned to the RP.
3) Fixed interconnections are used for connecting each RP in the design to the static logic. Early versions of the Xilinx DPR flow relayed on fixed interconnects called Bus-Macros (BMs) placed by the designer at the boundaries between the RPs and the static logic. The current DPR flow uses PROXY LUTs, which are basically 1-input LUTs, each of which is capable of routing one signal and is automatically inserted and locked in specific locations within the RMs.
4) RPs may only contain static route; no static logic is allowed inside the RPs. Furthermore, the PAR process does not allow overlapping RPs.

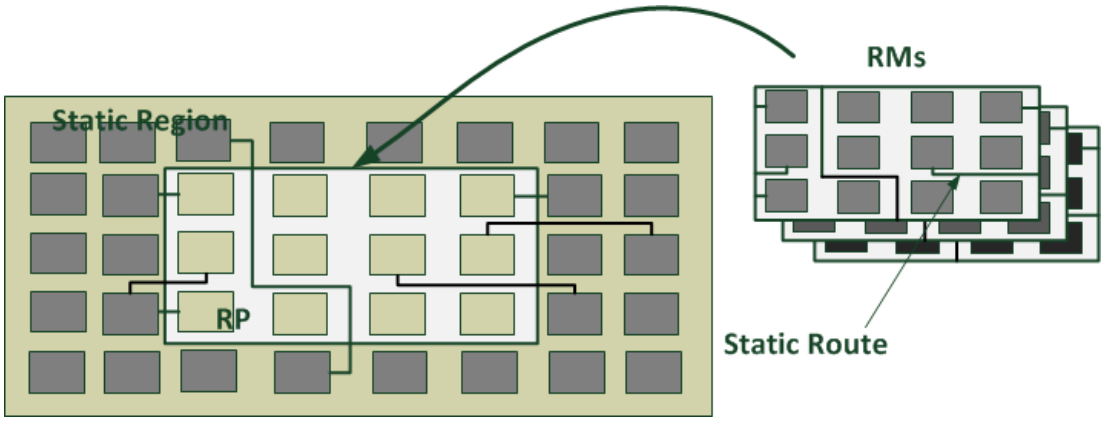



**Figure 2.10 Reserving static routes in modular DPR**

Not having control over the static routes in Xilinx DPR flow prevents some interesting architectures from being deployed efficiently in FPGAs. One example is the system shown in Figure 2.11 where several RPs of the same size are placed in the FPGA. Even if the same RM is assigned to all the RPs, each RP will require a different partial bitstream for the same RM. This increases the size of memory required for storing the partial bitstreams, especially when many RMs are assigned to the RPs (Figure 2.11a). Xilinx DPR flow also does not allow for the placement of overlapping RPs, which may lead to inefficient placement when large RMs are present in the design (Figure 2.11b).

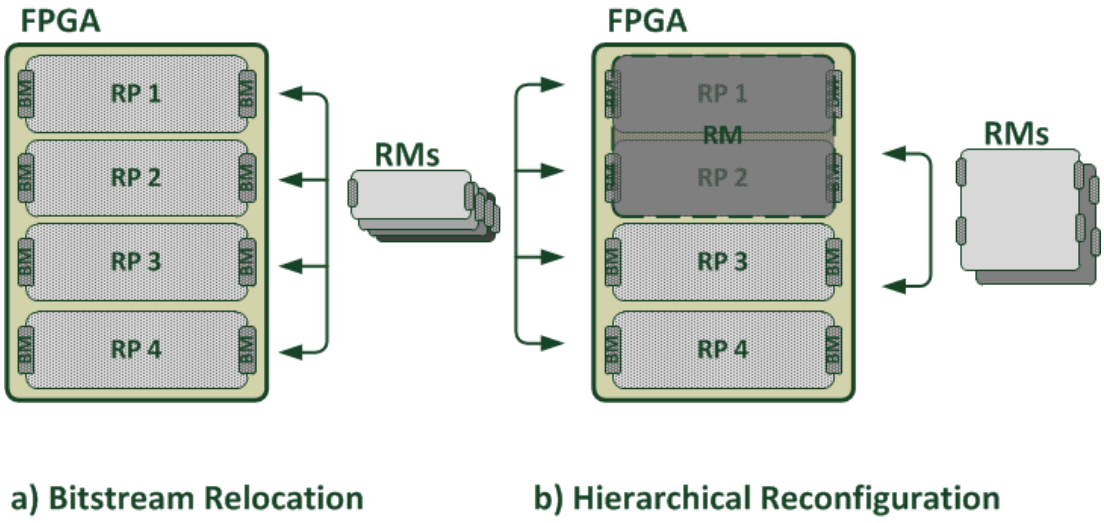



**Figure 2.11 Limitations of Xilinx DPR flow**

## 2.2.2 Altera DPR Flow

Altera is one of the major FPGA manufacturers and currently supports DPR in most of its new FPGA devices. The Altera DPR flow is fundamentally similar to the Xilinx DPR flow. Altera DPR flow starts with a base design containing a static region and at least one reconfigurable region. Similar to Xilinx DPR flow, a number of reconfigurable modules (called 'personas') can be assigned to each reconfigurable region. By having different revisions of the base design, each containing a different

set of personas, the Quartus software tool can generate the partial bitstreams of the design [30]. The first step carried out by the Quartus tool to generate the partial bitstreams in a DPR design is to compile all the revisions of the design to generate the Masked SRAM object Files (MSF) and the SRAM Object Files (SOF) for each revision (see Figure 2.12). In each revision of the design, an MSF file and an SOF file are created for each persona. These two files are used by the Quartus tool to generate a Partial-Masked SRAM object File (PMSF) for each persona before generating the partial bitstream files.

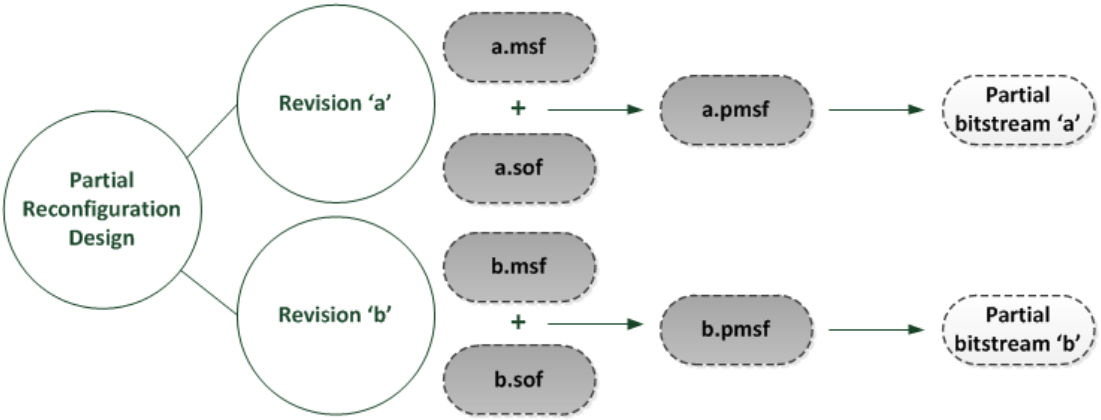



**Figure 2.12 Simplified Altera DPR flow**

### Limitations of Altera DPR Flow

Similar to Xilinx DPR flow, Altera DPR flow does not give designers control over the static routes and does not allow for bitstream relocation. While there is no technological limitation preventing bitstream relocation in Xilinx Virtex FPGAs as Xilinx Virtex FPGAs have regular routing structure, this is not the case for Altera FPGAs, which tend to have some routing variations and mismatches between what appear to be identical resources [31]. For this reason, all of the research work and academic DPR tools available in the literature, as well as the research work presented in this thesis are focused on bitstream relocation on Xilinx Virtex FPGAs.

### 2.2.3 Configuration Ports

Generally, DPR can be performed externally using one of the external configuration ports or internally using the ICAP. Table 2.1 shows the main configuration options available for Xilinx FPGAs. The fastest configuration port is the SelectMAP. This port allows for a configuration throughput of 400MB/s and it can be used for full device configuration as well as partial reconfiguration. However, the SelectMAP requires some additional external circuitries to control the configuration operation. The ICAP, on the other hand, provides an internal interface to the SelectMAP, which means that fast partial reconfiguration can be performed and controlled from within the FPGA by implementing the appropriate reconfiguration control logic. The ICAP provides full internal read and write access to the FPGA's configuration memory leading to the possibility of fully autonomous systems.

In Xilinx FPGAs there are two ICAPs; however, only one ICAP can be used at a time. The two ICAPs can be seen as a 2-to-1 multiplexer implemented on the SelectMAP interface. The active ICAP is referred to as the primary ICAP and the redundant ICAP is referred to as the secondary ICAP. Switching between the two ICAPs is possible at run-time by writing the appropriate switching commands through the primary ICAP and then switching operation to the secondary ICAP [25].

**Table 2.1 Configuration ports [25]**

| Configuration Port | Type | Max. Frequency (MHz) | Max. Data Width |
|---|---|---|---|
| ***JTAG*** | External | 66 | 1 |
| ***SelectMAP*** | External | 100 | 32 |
| ***ICAP*** | Internal | 100 | 32 |

### 2.2.4 Bitstream Relocation

Bitstream relocation is the ability to configure the same partial bitstream in different locations on the FPGA. Such partial bitstream is referred to as a relocatable partial bitstream, or a relocatable bitstream for short. Bitstream relocation is generally used

to reduce the memory size required for storing partial bitstreams when the same RM is instantiated in several locations in the FPGA.

There are several applications that can make use of bitstream relocation. The lack of official support from Xilinx for such a feature led to the development of several advanced tools and configuration techniques to circumvent the limitations of Xilinx DPR flow. In general, successful bitstream relocation has four main requirements, which are summarised in the following sub-sections.

**Resource Compatibility**

A partial bitstream configures a fixed height and width of resource columns. In order to relocate an RM, the target location of configuration must have identical resources to the original location of the RM. This implies that any target location of a relocatable bitstream must have the same dimensions, resource type and column layout.

The maximum number of feasible locations for a relocatable RM will depend on the number of regions with the same resource layout on the FPGA (see Figure 2.13). Modern FPGAs consist of a regular arrangement of specialised resource columns. In most cases, the column arrangement is not fully regular, as can be seen from Figure 2.13. This may limit the number of feasible locations for an RM, especially if the RM is large and spans different types of resource.

As the original layout of the RM influences the maximum number of feasible locations for a relocatable RM, more than one partial bitstream can be generated for the RM, each with a different resource arrangement, to expand the total number of feasible locations for the RM [32].

In some cases, a partial bitstream can be modified online to fit a region with a different resource layout. The authors in [33] demonstrate a successful technique for relocating an RM containing a DSP column to a target location containing a similar resource arrangement but with a BRAM column instead of the DSP column. This was possible because the DSP column was not used by the RM and the routing

through the DSP column was made identical to the routing through the BRAM column in the target location.

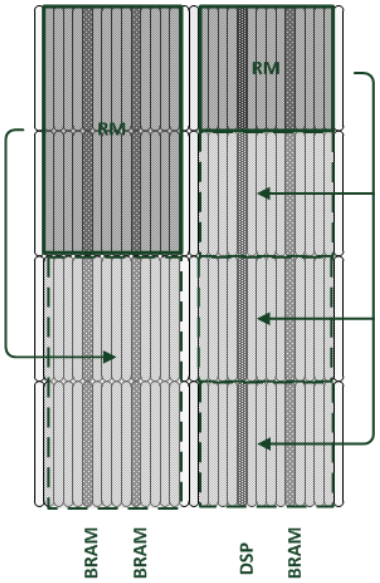



**Figure 2.13 Feasible RM relocation**

### Reserving the Static Routes

When relocating an RM to a target location, the static routing must be reserved and not corrupted by the reconfiguration process. As mentioned earlier, Xilinx DPR flow does not give designers control over the routing process. There are two main methods discussed in the literature to deal with the static routes while relocating RMs in different locations on the FPGA. The first method is based on reserving the routing in the target location by performing the necessary modifications to the relocatable bitstream, which will keep these routes intact after configuration. An example of this method is demonstrated in [34], where the configuration frames in the empty target location are read and XOR-ed with the relocatable bitstream to ensure that the routing configuration bits are reserved after configuration (see Figure 2.14a). This method only works if the relocatable RM does not use any of the routing resources used by the static logic in the target location. A similar method is presented in [35], where all the feasible locations of the RMs are pre-computed and special

files containing the routing configuration data are generated for each location. This accelerates the relocation as no configuration memory readback operations are required.

The second method for protecting the static routes is based on creating restricted regions on the FPGA that are free of static routes. As this is not possible using the standard Xilinx tools, workarounds have been proposed that are based on placing some blocking circuitries in the restricted regions to prohibit the PAR process from using routing resources in this region (e.g. [36] and [37]). These blocking circuitries consume all the routing resources in restricted regions and consequently force the router in the PAR process to use the routing resources outside these regions for the static routes (see Figure 2.14b). The OpenPR tool presented in [36] works alongside the Xilinx tools and uses the blocking technique to block all static routes from certain regions on the FPGA. The GoAhead tool presented in [37] uses a similar method; however, it gives the option of allowing some of the static routes in the reconfigurable region as long as they are not used by any RM in order to reduce congestion and latency.

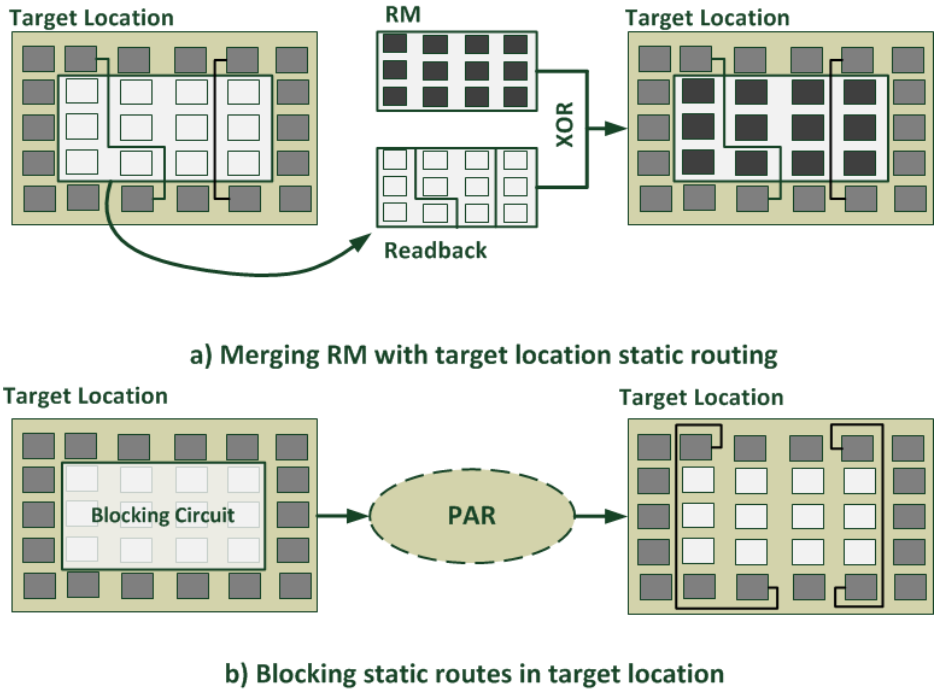



**Figure 2.14 Reserving static routes for RM relocation**

### Reserving RM Connections

When relocating RM to a target location in the FPGA, the connections between the RM and the other components in the system must be reserved. Theoretically, the routing process can be repeated after each relocation operation to re-establish the connections between the relocated RM and the system. However, this is impractical as knowledge of how the bitstream maps into the routing resources is required in order to implement an online router. In addition, the large routing time overhead due to the large number of routing resources in FPGAs will prevent efficient online routing. In [38], Shayani et al. propose using pre-compiled routing components based on the CLB resources, which can be tiled to form vertical and horizontal connections between two modules in the system. This reduces the routing problem as only a few routing components need to be used to form a connection between two modules. However, this method is inefficient when trying to connect widely separated components due to the large number of logic resources required for routing and the propagation delay caused by the long connections.

Most of the bitstream relocation systems proposed in the literature rely on fixed infrastructures of interconnects for connecting RMs rather than online routing. Bus Macros (BMs) provide pre-routed point-to-point connections and can be used as fixed interconnections for RMs when placed on specific locations on the boundaries between the RPs and the static logic (see Figure 2.15). Traditionally, BMs were based on Tristate Buffers (TBUFs), which where embedded in the early Xilinx FPGAs. LUT-based BMs replaced TBUFs after the Virtex-4 was introduced. As the current Xilinx DPR flow does not support BM integration, academic tools have emerged for the generation of custom BMs [39], and automatic placement of BMs ([37] and [40]).

Fixed interconnects can be used to build several on-chip communication architectures. The simplest architecture is the slot-based architecture wherein several slots with fixed interconnects are connected to a crossbar (see Figure 2.16a). RMs can be freely relocated between the slots and the crossbar can be programmed to establish the desired point-to-point connections [41]. Network-on-Chip (NoC)

topologies can be used as an alternative to the crossbar by implementing routers between the fixed interconnects [42] (see Figure 2.16b).

A bus-based architecture can also be used. In [43], Koch et al. present the ReCoBus tool, which allows for connecting several slots through a fixed horizontal bus (see Figure 2.16c). RMs can be configured on top of these slots and connected to the bus using special connection macros.

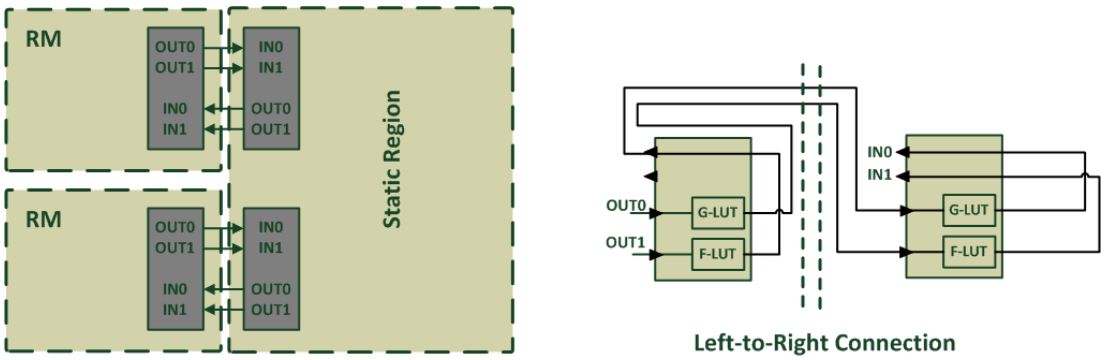



**Figure 2.15 LUT-based BMs**

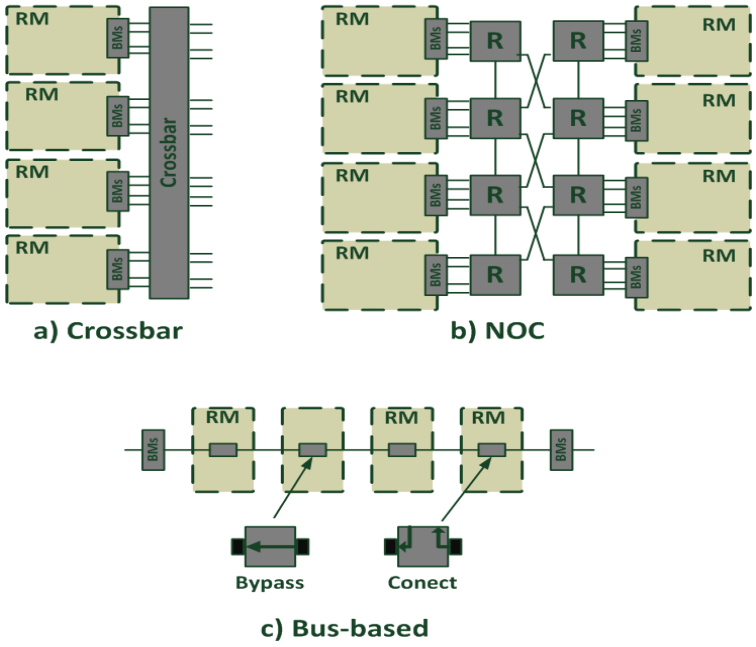



**Figure 2.16 On-chip communication infrastructures for relocatable RMs**

### Bitstream Manipulation

The configuration location of a bitstream can be altered by modifying the frame addresses in the bitstream. The frame addresses in a bitstream specify the physical locations of the configuration frames on the FPGA fabric. If a bitstream is required to be relocated to a target location, the frame addresses for the target location must be identified to replace the original frame addresses. Bitstream modification can be performed externally prior to configuration of the relocatable RMs or internally using dedicated logic implemented in the FPGA. Early bitstream relocation systems such as BITPOS and PARBIT relied on external processors to carry out the bitstream modifications ([44] and [45]).

Some systems allow for bitstream relocation to be performed internally using a processor implemented on the FPGA's logic ([46] and [47]). In these systems, the processor scans the bitstreams file stored externally and modifies the addresses field before configuration through the ICAP.

For systems requiring fast internal bitstream manipulation, bitstream filters are proposed to accelerate the relocation process. A bitstream filter is a dedicated configuration controller that automatically filters and modifies the address fields in the bitstream when it is streamed for configuration. REPLICA was an early bitstream filter designed for CLB-based cores relocation through the SelectMAP interface in Virtex-2 FPGAs [48]. The REPLICA2Pro was later introduced to support BRAM relocation as well as configuration through the ICAP in its 8-bit configuration mode [49]. The BiRF is another bitstream relocation filter introduced for newer FPGA devices, and it supports configuration through the ICAP in its 32-bit configuration mode [50].

## 2.3 Chapter Conclusion

This chapter introduced the Xilinx tiled-based architecture, which is the dominant architecture in current SRAM FPGAs. A brief introduction of the most relevant Xilinx reconfigurable resources was presented. This chapter also presented the Xilinx

DPR flow and discussed its limitations with focus on bitstream relocation, which is not supported in the Xilinx DPR flow. A discussion of the techniques proposed in the literature to enable bitstream relocation was also presented in this chapter.

# Chapter 3 : Dynamic Partial Reconfiguration for High Performance and Reliability

High Performance Computing (HPC) involves the use of parallel processing techniques to solve large and complex computational problems. Until the early 2000s, single-core CPU systems were the mainstream choice for HPC applications due to their low cost compared to supercomputing architectures. CPU performance and frequency continued to scale in line with Moore's law until the Mid-2000s, when the trend of multi-core CPU architectures started to take over to meet high-performance demands. Recently, new architectures involving the use of hardware accelerators as co-processors are emerging as an alternative to CPU-only systems. This has opened the door for acceleration devices such as FPGAs and GPU to play a key role in the advancements of HPC. FPGAs in particular are very interesting prospects for High Performance Reconfigurable Computing (HPRC) applications as they offer a great level of flexibility without compromising on performance.

The flexibility of FPGAs also opens the door for implementing interesting adaptive Fault-Tolerant (FT) systems. An FT system is a term given to a system that is specifically designed to prevent failure in system operation when one or more faults occur in some of the system's components. The development of FT system is a major interest for researchers in different disciplines, covering a wide range of applications in space, aviation and military. Adaptive Computing Systems (ACSs) depend on reconfigurable platforms to adapt their behaviour to changes in the external environment. ACSs are often deployed in hostile environments under harsh conditions, such as high levels of radiation and extreme temperatures, making system upgrade and repair difficult and costly. The cost of repair in such environments increases the demand of reliable and easily upgradable hardware.

# 3.1 DPR Deployment in High-Performance Systems

The possible gain in performance achieved in FPGAs can be enormous compared to other computing platforms. The true power of FPGAs comes from the computational parallelism that can be achieved using the available hardware resources to handle a given problem. With the continuous increase in device density and decrease in power consumption in every device generation, FPGAs are attracting the attention of researchers as a high-performance solution for several applications. The reprogrammability and flexibility of FPGAs makes them a favourable choice for engineers who require constant modifications to their designs during the development stage or in the field. In modern SRAM-based FPGAs, the system can be reconfigured fully or partially to alter the computation functionality at runtime. Run-time reconfiguration can enhance performance [21], and increase functional density [22]. In general, the increase in performance brought by run-time reconfiguration comes from achieving more execution parallelism through optimal exploitation of the FPGA resources.

DPR further enhances the device flexibility by allowing changes to the functionality of certain functional blocks without stopping the system. This opens the door for new reconfigurable platforms architectures for hardware acceleration in FPGAs where multiple customised accelerator cores can be swapped in/out of the FPGA on demand. With the enhancements seen in recent embedded processors such as the Xilinx MicroBlaze soft-processor [51] and the PowerPC hard-processor [52], the FPGA can be configured as a standalone system that schedules and allocates its own payload of tasks and handles the reconfiguration operations internally. In order to achieve the desired gain in performance when using DPR, the internal reconfiguration time overhead must be minimal.

## 3.1.1 FPGA-based Acceleration in HPC

In [53], Xilinx has classified the current trends of FPGA deployment in HPC applications into three categories: connectivity bridging, fixed function hardware

acceleration and software acceleration. In the first category, FPGAs are used as bridges and switches for interfacing different subsystems. The flexibility of FPGAs allows designers to make changes to their design to accommodate for any changes in the IO requirements. When used as fixed hardware accelerators, FPGAs are used to accelerate a fixed function which requires high processing throughput by implementing a hardware accelerator that processes the data in parallel. Using FPGAs for software acceleration is based on moving portions of the processing usually performed by CPUs to an FPGA co-processor. This type of acceleration is particularly interesting as it allows for the creation of generic computing platforms that can be used in different applications. While traditional FPGA acceleration platforms are based on connecting a single FPGA or a cluster of FPGAs to a CPU over Ethernet or PCIe, recently the industry has seen a shift towards CPU/FPGA hybrid SoCs aimed at high-performance embedded computers. The Zynq-7000 from Xilinx [54] and the Altera SoC FPGAs [55] are recent SoCs provided by the two main FPGA vendors. Both use a dual core ARM Cortex-A9 processor combined with their latest FPGA technology.

One example demonstrating the potential of FPGAs in HPC is the implementation of reconfigurable systolic array accelerators. Systolic arrays were first proposed by Kung in 1982 [56]. Systolic arrays are a grid-like structure of special Processing Elements (PEs) that process data in a pipelined fashion to achieve a high level of parallel processing, making them very suitable for computationally intensive operations. The name ‘systolic’ is derived from the Latin term ‘systole’, which is a medical term used to describe the regular pumping of blood by the heart. The name ‘systolic’ was coined from the medical terminology because the propagation of data into the systolic array resembles the propagation of blood in the human circuitry system, and the operation of PEs which process the data and injects partial results into the data stream resembles the operation of the organs in the body. In [57], Johnson differentiates between general purpose systolic array architectures and customized systolic arrays, which tend to have better performance but the lack the flexibility required to implement different algorithms using the same hardware. Johnson also emphasises the importance of reconfigurable systolic arrays in FPGAs

that offer a high level of customisation without compromising flexibility. Successful FPGA implementations of various systolic array accelerators have been reported in the literature for several applications in bioinformatics ([58] and [59]), DSP ([60] and[61]) and data mining of large databases ([62] and [63]).

There are attempts to automate the generation of the RTL-level code for the systolic array accelerators in FPGAs using specialised software tools. In ([64] and [65]), the authors presented the ROCCC tool, which is a C-to-VHDL compiler tool capable of generating optimised systolic array accelerators for several applications. In [64], the authors demonstrate their tool to accelerate the Smith-Waterman algorithm, which is widely used for local and global sequence alignment in bioinformatics [66]. They implemented a software-generated systolic array in the SGI RASC RC100, which contains two Virtex-4 LX200 FPGAs and connects to an SGI server. The performance gain achieved was over 300x compared to a 2.8 GHz Intel Xeon CPU. A build up work to the ROCCC compiler was presented in [67], where the authors presented a tool that generates a complete FPGA implementation for perfect nested loops that support off-chip DRAM memory access. Another related work is the LegUp tool, which generates a complete FPGA implementation from a C code. The system generated by the tool consists of a Tiger MIPS soft-processor and custom hardware accelerators that communicate with the CPU using a standard bus [68]. Opposite to hard-processors, which are hardwired prefabricated processors; soft-processors are designed using the standard FPGA design flow to be implemented on the FPGA logic. As the LegUp tool uses a soft-processor for running the software part of the system, the entire system can be implemented in the FPGA fabric. This allows for a single-chip solution of a hybrid system without the need for a specialized SoC, which contains an integrated ASIC-processor, making the technology applicable to a wider range of FPGA families.

In order to achieve a higher performance and a higher flexibility in a hardware/software hybrid system, DPR can be deployed to control the type and number of active accelerators during the operation of the system. In [69], the authors propose a framework for systolic array acceleration in FPGAs, which contains an embedded soft-processor and multiple reconfigurable regions defined as ‘sockets’

dedicated for placing the systolic array accelerators. The proposed system allows for accelerating two algorithms running concurrently by deploying DPR to alter the size of the systolic array assigned for each algorithm in a given time. Each socket contains a BM that is connected to a switch box. The switch box can be controlled by the processor to connect several sockets together. As each systolic array is customised for accelerating a particular algorithm, the level of acceleration for a running algorithm can be changed by altering the number of sockets assigned for its systolic array (Figure 3.1a). Relocatable partial bitstreams for the systolic arrays are proposed to reduce the storage memory requirements of the system. In the system demonstrated in [69], the size of each partial bitstream is determined by the size of the slot. A similar system for systolic array acceleration is proposed in [70], where the authors aim to further reduce the memory required for storing the systolic array partial bitstreams by having smaller relocatable partial bitstreams that can be concatenated horizontally within each socket (Figure 3.1b). Although this approach will reduce the storage memory requirements for the system by reducing the granularity of relocation, it suffers from several flaws not discussed by the authors that can lead to degradation in system performance. Reducing the size of the relocatable partial bitstream will make routing across each socket more difficult, especially when a large bus is required to feed the PEs in the systolic array. In addition, smaller reconfigurable regions will have less resource-packing efficiency. In Xilinx FPGAs, the average packing density possible for a reconfigurable region is around 80% [29]. For relocatable partial bitstreams, the packing density can be smaller as some resources are used for the dedicated routing and BMs. When a number of these relocatable partial bitstreams are concatenated together to form a large systolic array, the total number of PEs will be reduced compared to a single partial bitstream covering the same area.

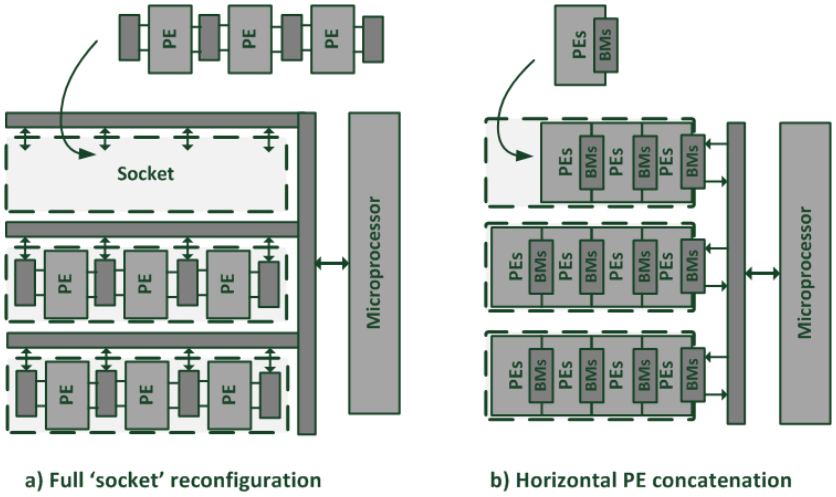



**Figure 3.1 DPR-based systolic array acceleration**

Software acceleration with DPR can achieve higher performance than static accelerators by exploiting more parallelism [71]. Figure 3.2 shows two implementations of a system that utilises three hardware accelerators activated one after the other. The first implementation is a static implementation, so the available resources are divided between the three accelerators. The second implementation is based on DPR, so the accelerators can share the available resources, thus allowing for larger accelerators with shorter execution time.

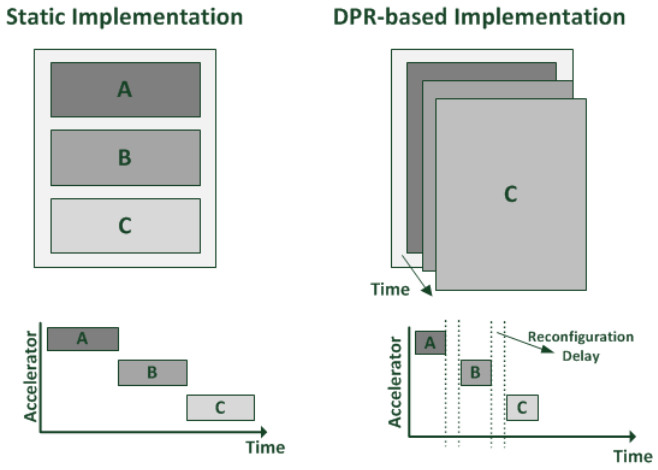



**Figure 3.2 Enhanced software acceleration with DPR [71]**

### 3.1.2 Reconfigurable Operating Systems

The idea of an ROS was first proposed by Brebner in [12], where an OS is proposed to manage the execution of tasks using the reconfigurable hardware. An ROS is supposed to hide the complexity in mapping these tasks into the available hardware resources from the user and enable high-level programming of reconfigurable applications. According to Brebner, any task executed on the reconfigurable hardware is defined as a Hardware Task (HT). There are several characteristics that differentiate an HT from a software task [72]. These most important characteristics are summarised by Table 3.1. An ROS has two main advantages over a normal OS, which allow for higher system performance. The first is that the HT's hardware can be tailored to the needs of the task and designed with a high level of parallelism. This flexibility in hardware customisation can make HTs considerably faster than their software counterparts. In addition, the number of tasks that can run concurrently in an ROS depends on the area of the reconfigurable fabric rather than the number of fixed CPUs allowing for better true multitasking as smaller tasks only consume small areas of the reconfigurable fabric.

**Table 3.1 Characteristics of hardware and software tasks [72]**

| Characteristic | Software Task | Hardware Task |
|---|---|---|
| ***Design*** | Software programming language (e.g., C, C++, assembly) | HDL description or C-to-silicon programming |
| ***Executing Device*** | Processor | FPGA |
| ***Executing Unit*** | CPU | Custom relocatable module |
| ***Execution Nature*** | Sequential | Parallel ( depends on module design) |
| ***Execution Time*** | Depends on CPU clock speed | Depends on design and module clock speed |
| ***Maximum Task Parallelism*** | Depends on number of CPUs | Depends on FPGA's area |

In [73], Wigley et al. describe the main practical challenges in implementing an ROS in FPGAs. The authors also define how task allocation and scheduling should be approached in an ROS.

### Task Allocation

Task allocation is the process of assigning available resources on the reconfigurable hardware for task execution. In FPGAs, HTs can be seen as pre-compiled relocatable partial bitstreams, which are reconfigured at run-time to execute a given function. When HTs are assumed to be rectangles with fixed heights and widths and the FPGA is assumed to be a large uniform area of logic resources, the allocation of HTs can be seen as a 2-D packing problem where the FPGA is partitioned into smaller areas used for the placement of the HTs. In [73], Wigley et al. suggest that a task allocation algorithm should reduce chip fragmentation. As chip fragmentation could create several 'dead' regions not suitable for the placement of any tasks, tasks should be packed as close as possible to each other to expand the free space on the chip and increase the number of tasks that can be allocated in a given time of operation.

There are several approaches discussed in the literature to reduce chip fragmentation in an ROS. In [74], Bazargan proposes partitioning the FPGA area into overlapping empty rectangles with the objective of Keeping track of All Maximum Empty Rectangles (KAMER). Bazargan also proposes Keeping track of Non-overlapping Empty Rectangles (KNER). In both schemes, the FPGA area is scanned to determine all possible Maximum Empty Rectangles (MERs). When an HT is required to be allocated, a scan through all the MERs is performed to find a suitable location. An area-fitting algorithm, such as First-Fit (FF) and Best-Fit (BF), is used to select a suitable location for the HT. The FF algorithm scans the empty rectangles and selects the first rectangle capable of fitting the HT, whereas the BF algorithm selects the empty rectangle with minimal difference in area compared to the HT. Once an area is selected for an HT, the FPGA area is scanned again and the MERs are updated.

To simplify the MER scanning process, Walder et al. propose the use of a hash matrix, which contains pointers to a list of MERs with the same area [75]. In order to reduce the time for allocating consecutive tasks, they propose to update the hash matrix while each task is allocated. Morandi et al. present a related work in [76], where the FPGA area is transposed into a tree structure with nodes representing

occupied areas and leaves representing MERs. Using this tree structure only leaves need to be scanned for HT allocation.

In contrast to MER-based allocation, other works base the allocation process on keeping track of the Vertex List Set (VLS), which indicates the positions of placed tasks [77]. The proposed algorithm allocates tasks in positions with the highest contact length with neighbouring placed tasks or the left-side edge of the FPGA. A similar work proposes keeping track of the occupied area rather than the empty area in the FPGA [78]. The proposed allocation scheme scans the FPGA to find the Impossible Placement Region (IPR). Tasks are then allocated in the nearest optimal position to the IPR. Figure 3.3 illustrates the main 2-D allocation algorithms.

**Task Scheduling**

Task scheduling is the process of determining the order of execution of the consecutive tasks where tasks are assigned with specific priorities. Usually, priorities are assigned according to the tasks execution deadlines. The execution deadline is defined as the maximum delay for a given task to finish its execution and generate its results. The task priorities can be fixed, as seen in the Deadline Monotonic (DM) scheduling where tasks with the shortest relative deadline are assigned with the highest priorities [79]. Task priorities can also be assigned dynamically by assigning the highest priorities to tasks with the nearest deadline as seen in the Earliest Deadline First (EDF) scheduling [80]. Scheduling can be pre-emptive, which enables higher priority tasks to stop the execution of the lower priority tasks and start its execution. Non-preemptive scheduling on the other hand does not allow high priority tasks to interrupt currently executing tasks.

In order to make task scheduling applicable to an ROS implemented on an FPGA, the reconfiguration port delay time must be considered. In current FPGAs only a single configuration port can be active at a time to carry out the reconfiguration operations. With the sequential nature of dynamic reconfiguration and the fixed bandwidth of the reconfiguration port, access to the reconfiguration port must be shared among different operations. In [81], the proposed system schedules access to the reconfiguration port according to the task deadline using conventional scheduling

algorithms (DM and EDF). The authors in [82] suggest that already placed tasks that have finished their current execution should be considered for the execution of future tasks to reduce the overall reconfiguration time. Other work also considers the communication time required for each scheduled HT ([83] and [84]).

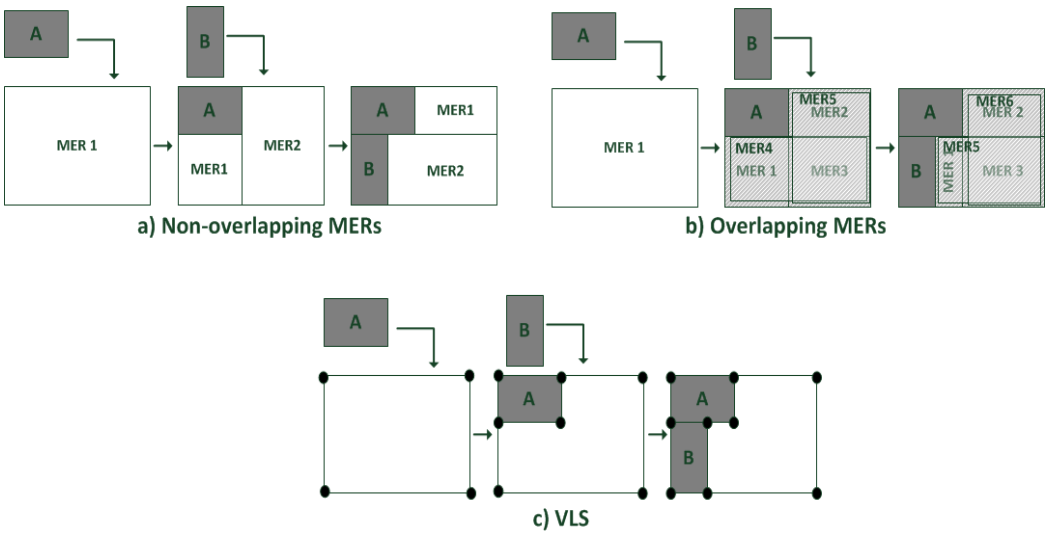



**Figure 3.3 2-D task allocation algorithms [85]**

### 3.1.3 Reducing Reconfiguration Delay

The efficiency and speed of an internal reconfiguration controller is critical in high-performance embedded systems, especially for systems that extensively use the ICAP for different configuration operations. An example of this is an ROS kernel that uses the ICAP for the persistent HT allocation and de-allocation. The configuration time overhead is a performance bottleneck in such systems, especially with the sequential nature of configuration in current SRAM-based FPGAs, which do not allow for multiple reconfiguration operations to run concurrently.

There are several techniques proposed in the literature to accelerate the configuration process in order to meet the demand for high-performance systems. Generally, accelerating internal configuration can be achieved by:

### Accelerating Bitstream Fetch-Time

SRAM-FPGAs are volatile, full and partial bitstreams are initially stored in a non-volatile memory. Typical non-volatile memory modules have high latency and low throughput, and they are not suitable for high speed configuration. A common practice for high-speed run-time reconfiguration is to store a copy of the required partial bitstreams in a faster SRAM or a DRAM memory module after power-up of the device. Typically, ICAP controllers are designed as slave cores that connect to master CPU through a standard bus. An example of such a controller is the Xilinx HWICAP IP core which connects to a Microblaze or a PowerPC processor. Early versions of the ICAP controller use the PLB bus while more recent versions use the AXI bus for connection with the master processor ([86] and [87]). In its basic configuration, the HWICAP depends on the master CPU for controlling the streaming of data from external memory to the controller's internal buffers. This configuration is inefficient for large partial bitstreams as the CPU will be constantly busy with loading the controller's internal buffers and not able to carry out other tasks during the configuration process. In addition, the software overhead for initiating the different data transfer requests through the PLB bus is large, resulting in poor configuration throughputs. In [88], Liu et al. investigated the maximum throughput achieved with the HWICAP core in different modes. When connecting the HWICAP as a slave device to OPB/PLB bus, the average throughput achieved was in the range of 0.61-19.1 MB/s, which is well below the maximum theoretical throughput of the ICAP of 400MB/s [86].

Direct Memory Access (DMA) has also been considered to accelerate the data transfer from external memory where a DMA controller is responsible for fetching data from the external memory via the PLB bus. A DMA controller has been applied to the basic HWIAP configuration in [88], where the processor is only responsible for initiating the data transfer by instructing the DMA controller to perform a burst transfer from memory to the internal buffers of the controller. This was shown to increase the throughput to 82.6 MB/s, which is still far from the optimal configuration throughput. The modest improvement was mainly because the design did not account for the latency of each burst data transfer. Other research work has

reported throughputs approaching the maximal configuration throughput using DMA with custom designed ICAP controllers. The controller presented in [89] uses a large burst length and a large word length (256 bit) to eliminate the impact of latency on the throughput. The achieved throughput approached the maximum throughput; however, 8 BRAMs were used for an asynchronous FIFO to translate the 256-bit word length to the 32-bit of the ICAP.

There are also other controllers that do not rely on a DMA controller for data transfer through the PLB bus. An example is presented in [90], where a processor feeds the ICAP with configuration data through a Fast Simplex Link (FSL). The aim of this work was to achieve acceptable performance with a lightweight controller, which can be easily reused in different designs. Another example considers using the Xilinx Native Peripheral Interconnect (NPI), which is the fastest connection for the Multi-Port Memory Controller (MPMC). The proposed controller used two ports of the MPMC for the ICAP control, which supports configuration readback in addition to bitstream configuration [91].

The partial bitstreams can also be stored in on-chip memory blocks to allow for the shortest latency possible ([88] and [92]). Although using on-chip BRAM blocks would allow for fast data transfer, only storing small partial bitstreams would be possible, making this method impractical.

**Bitstream Compression**

Generally, bitstream compression is deployed to reduce the storage memory required to store different bitstreams. Compressed partial bitstreams will require a de-compressor implemented in the FPGA logic to restore the configuration data to its original content. Bitstream compression can reduce the overall configuration time by reducing the bitstream fetch-time from slow external memory devices. In [93], Koch et al. explored different compression algorithms and showed that with bitstream compression the maximum configuration throughput of 400MB/s can be achieved with storage devices supporting only half the required bandwidth. In [94], Liu et al. explored the natural redundancy in Xilinx FPGA's bitstream to come up with a simple decompression scheme that does not require a large decompression circuit

implemented in the FPGA logic. The presented compression scheme is based on finding repetitions of consecutive configuration words; these words are then encoded into a smaller segment consisting of two words, one containing a configuration value and the other containing the number of repetitions of this value. They also considered removing the padding words and the No-Operation (NOP) commands from bitstreams and adding them on-line during the configuration process. The compression ratio achieved with their encoding scheme was in the range of 1.09-3.15 and the maximum improvement in configuration time was around 17%.

Another method to enhance the configuration time through bitstream compression is by using the Multiple Frame Write (MFW) feature in Xilinx FPGAs. The MFW feature allows for writing multiple configuration frames containing the same content once instead of writing them individually [25]. This compression feature is integrated with the Xilinx FPGA's internal configuration circuitry so the reduction in bitstream size is directly proportional to the increase in configuration time. In addition, it does not require padding for writing frames to the configuration memory. The Combitgen tool presented in [95], is based on manipulating different configurations of a particular reconfigurable module and extracting the similarities and differences in their partial bitstreams. The tool then generates smaller configurations that consist of the configuration frames required to achieve a context switch from the implemented top-level module. These frames are configured using the MFW feature to further reduce the reconfiguration time.

**Overclocking the ICAP**

The maximum clock frequency rated for the ICAP in current Xilinx FPGAs is 100MHz. The ICAP supports a write width of up to 32-bits, giving a maximal theoretical reconfiguration throughput of 400MB/s. Several authors have reported successful reconfiguration with overclocking. An example can be seen in the METAWIRE on-chip communication system [96], where the authors have implemented a custom ICAP controller to transfer data between different buffers in the system to emulate the operation of an NoC. The reported maximum clock frequency was 144 MHz in a Virtex-4 FPGA. In [97], a higher clock frequency of

200 MHz was achieved in a Virtex-5 FPGA. In [92], Hansen et al. report a much higher ICAP clock frequency of up to 533 MHz in a Virtex-4 FPGA. Unfortunately, no information was provided on the reliability of their controller when operating at such high frequencies. In addition, they assume that the test partial bitstream are stored in BRAM blocks placed as close as possible to the ICAP, which is not practical for real applications.

The maximum clock frequency of the ICAP controller will be affected by several factors such as the speed grade of the device, the routing and placement of the full implemented design and other environmental variables such as temperature. It is difficult to predict the behaviour of an over-clocked ICAP across different designs and under different conditions. To address this problem, Hoffman et al. propose an active feedback monitoring circuit, which generates an optimum clock based on voltage and temperature measurements [98].

**RM Prefetching**

In systems deploying several RMs, it is possible to configure an RM before it is scheduled for execution while other RMs are still executing. This way, its configuration delay will be overlapped with the other RM execution time. Figure 3.4 shows a system deploying four RMs that are required to be executed one after the other. Using two RPs, an RM can be pre-loaded into an RP while the preceding RM is still executing in the RP. This can greatly reduce the effect of the reconfiguration delay on the overall execution time of the system.

Prefetching can also be deployed in more complex systems, such as an ROS implemented on an FPGA. However, the order of HT execution is not deterministic in an ROS. Prediction algorithms can be deployed in an ROS to predict which HTs would result in the best performance gain when pre-fetched [99].

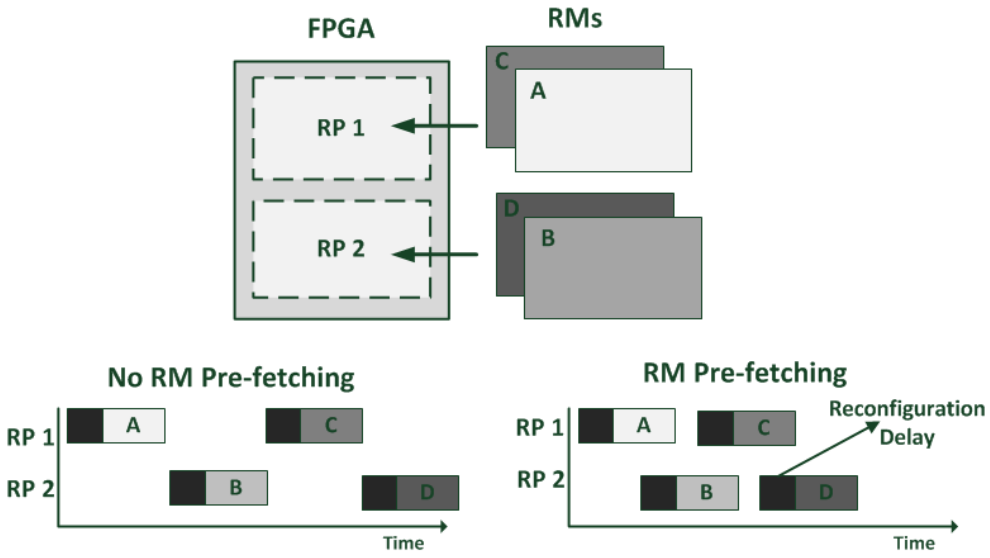



**Figure 3.4 RM pre-fetching**

## 3.2 DPR for Enhanced Fault-Tolerance

FPGAs are inherently flexible, making them an ideal platform for ACSs and an interesting prospect for space applications. Currently, SRAM is the most common technology for FPGA configuration due to its ease of fabrication and reprogrammability. However, SRAM technology is known to be sensitive to radiation-induced faults. In addition, faults in the configuration memory of FPGAs are not simply faults in raw data, which is stored in memory; they are transposed to the functionality and hardware structure of the implemented system leading to a complex impact on the system's behavior. This derives the need for innovative solutions to realise the full potential of FPGAs in FT systems. Much of the research aimed at enhancing the reliability of FPGAs is based on the DPR, which allows for reconfiguring faulty blocks in the design at run-time without stopping the operation of the system.

### 3.2.1 Background on Faults in SRAM-FPGAs

In semiconductor devices, faults can be divided into three main categories: Permanent, Intermittent and Transient [100]. Permanent faults, also known as hard faults, manifest themselves as irreversible physical defects in the device. There are several factors and physical effects that lead to permanent faults in semiconductor devices. Electromigration occurs when the collision of electrons with the metal atoms cause gradual movement of the ions in the conductor. The moving ions can accumulate or deplete in some regions of the conductor, causing short- or open-circuit faults. Electromigration is highly affected by the type of conducting material used. Since the adoption of copper interconnects in the semiconductor industry, the rate of permanent fault has dropped due to the high electromigration threshold of copper compared to aluminium. The Hot Carrier Injection (HCI) phenomenon can also contribute to the degradation of VLSI circuits by changing the switching characteristics of CMOS transistors, leading to delay faults. HCI can gradually cause a build-up of charges that gain sufficient energy to get trapped in the gate-channel interface of the transistor leading to reduced mobility and increased threshold voltage [101]. The Dielectric Breakdown (DB) phenomenon can cause what is normally an insulator to conduct electricity at a high electric field. In transistors, DB can cause an increased leakage current at the gate of the transistor which eventually leads to a short circuit ([102] and [103]).

Intermittent faults are faults that repeatedly occur at the same location as a result of physical instability to environmental changes such as temperature and voltage. Intermittent faults usually cause burst errors in the affected location; it is common for intermittent faults to appear before the occurrence of permanent faults. Errors induced by intermittent faults can be confused with transients, also known as soft errors. Transient faults are temporary errors that can be triggered by several factors such as exposure to alpha particles and cosmic ray neutrons, power supply and interconnect noise, electromagnetic interference and electrostatic discharge [100]. Radiation-induced soft errors are particularly important in spacecraft and aviation electronics. They can appear as glitches in logic, in this case called Single Event

Transient (SET), or bit-flips in memory cells and registers. Bit flips, also called upsets, can appear as Single Event Upsets (SEUs) or as Multiple Bit Upsets (MBUs). MBUs occur when a single radiation event flips multiple bits in storage circuits [104]. SRAM technology is especially sensitive to radiation-induced soft errors because the critical charge required to cause a bit-flip is relatively small. In an early study on soft errors in SRAM, it was shown that an SRAM chip supporting many megabytes of storage, can exhibit a Soft Error Rate (SER) that exceeds 50,000 FIT (failure per $10^9$ hours of system operation) [104]. This approximately translates to one error every two years. In another study by Tezzaron Semiconductor, it was reported that the average SER in an SRAM chip is between 1,000 to 5,000 FIT/Mbit [105]. In addition, the hard errors caused by particles with high energy are estimated to be 2% of the total errors. Although these error rates might be acceptable for some applications, they cannot be acceptable in FT systems, especially with the continuous increase in density and shrink in device geometry in SRAM, leading to higher SERs in every generation [106].

In FPGAs, faults can appear in the configuration memory or in the other hardware components. In modern FPGAs, the routing accounts for most of the configuration memory. Faults in the routing bits of the configuration memory could have complex effects in the implemented design. However, not all the logic and routing resources are used in a particular implementation in an FPGA device. In addition, not all soft errors in the used resources will cause functional errors. Xilinx use the Device Vulnerability Factor (DVF) to estimate how much a particular design is susceptible to functional errors in their devices. According to the Xilinx 2013 reliability report, one in 20 upsets on average will cause a functional error in a typical design. In the worst reported case, one in 10 upsets will cause a functional error [107]. In the same report, Xilinx reported the error rates in their devices from data collected from the Rosetta experiment [108]. According to the report, a Virtex-4 FPGA is susceptible to 263 FIT/Mb in configuration memory and 484 FIT/Mb in Block RAM memory. (1 FIT = 1 upset per $10^9$ hrs).

With continuous process technology scaling, MBUs are also becoming more of an issue in FPGAs. MBUs are not only caused by high energy particles; some SEUs in

the routing of the FPGA will cause a bit-flip in different bits in the configuration memory [109]. A study carried out by the Jet Propulsion Lab (JPL) showed that MBUs are nearly three times more likely to occur in Virtex-4 FPGAs than Virtex-2 FPGAs and 27-33 times more likely to occur in a Virtex-2 FPGA than earlier Virtex FPGAs [110].

### 3.2.2 Reliability Features in Modern FPGAs

FPGA manufacturers usually offer radiation-hardened versions of some of their product families, such as the Virtex-4QV and the Virtex-5QV from Xilinx. These products provide better SEU tolerance; however, they cost much more than the commercial FPGAs.

In commercial FPGAs, parity bits are usually added to each configuration frame in the configuration memory. These parity bits are used for the detection/correction of bit-flips in the configuration memory. In the Xilinx Virtex-4 FPGA there are 12 Error Correction Code (ECC) parity bits located in the 21$^{st}$ word of each configuration frame. These parity bits are generated by the BitGen tool to detect bit-flips in the configuration memory. The detection/correction process using the parity bits embedded in the configuration memory requires extra user logic implemented on the FPGA fabric. Xilinx provides the Soft Error Mitigation (SEM) IP core for its Virtex-6 and 7-Series FPGA families. The SEM IP core enables automatic detection/correction of faults in the configuration memory. It also extends the bit-flip correction capabilities of the device by adding a Cyclic Redundancy Check (CRC) generator, which stores reference CRC values in internal BRAMs [111].

Other FPGAs use other methods for error detection; for example the Altera Startrix-5 FPFA uses a 32-bit CRC value for each configuration frame to allow for better detection/correction. The CRC is also used for configuration verification. When a bitstream is configured, pre-computed frame CRC values are compared with CRC values generated by an internal circuitry to determine if any fault has occurred during the configuration process. CRC is also used for configuration verification in Xilinx FPGAs; however, instead of using a CRC value for each configuration frame, a

single CRC value is used for the entire bitstream. Table 3.2 lists some of the FPGA devices along with their embedded SEU mitigation features.

**Table 3.2 Soft-error detection/correction capabilities in different FPGAs**

| Device | Frame Parity Bits | Description |
|---|---|---|
| ***Xilinx Virtex-4*** | 12 Hamming | Can be used to correct single-bit errors and detect double-bit errors in a configuration frame |
| ***Xilinx Virtex-6*** | 13 Hamming | Can be used to correct single-bit errors and detect double-bit errors in a configuration frame, with SEM IP double-bit errors correction supported |
| ***Altera Startix-4*** | 16 CRC | Can be used to detect single-bit, double-bit and three-bit errors in a configuration frame. Can be used to correct all single-bit errors and 99% of double-bit errors. |
| ***Altera Startix-5*** | 32 CRC | Detection: single-, double-, triple-, quadruple-, quintuple-bit errors: Correction: single-bit and double-bit errors. |

## 3.2.3 DPR Techniques for Enhanced Fault-Tolerance

### Soft Error Mitigation

Configuration memory scrubbing is one of the most common methods used for soft error detection and correction in the FPGA's configuration memory. There are two types of configuration memory scrubbing technique widely discussed in the literature: the first technique is referred to as 'internal scrubbing'. This technique is performed using internal components inside the FPGA chip without the aid of any external components. Usually, internal scrubbing utilises the parity bits and the embedded detection units in the FPGA, where the correction process is performed in three main steps. First, a configuration memory frame is read using the internal configuration port and stored in a dedicated memory block. After that, the embedded parity bits in the configuration frame are used to detect possible bit-flips in the frame. If the location of the fault is identifiable by the parity algorithm, the corrupted bit is flipped in the memory before writing the frame back to the configuration memory.

There are three main weaknesses in the internal scrubbing scheme. The first is that the correction capabilities are limited to those supported by the frame parity bits. The second is that not all the resources in the FPGA are covered by the scrubbing scheme; resources configured as dynamic memory elements are masked during scrubbing [112]. The third and most important drawback is that the internal control logic of any scrubber is susceptible to soft errors, which can lead to complete failure and in more severe cases to injecting extra faults in the system. This issue in internal scrubbers was addressed in [113] wherein the author demonstrated the use of Triple-Modular Redundancy (TMR) to protect a Virtex-4 internal scrubber from soft errors.

The second type of configuration memory scrubbing is 'external scrubbing', which does not use the parity bits embedded in the configuration frames; instead a reference bitstream stored in external non-volatile memory is used. The reference bitstream, also called the 'golden bitstream', can be used for comparison with configuration memory readback results. In this case the scrubbing scheme is referred to as 'read and compare', or it can be configured periodically to overwrite any possible faults in the configuration memory without the need for any kind of detection; this scrubbing scheme is commonly referred to as 'blind scrubbing'. The main advantage of external scrubbing over internal scrubbing is that the correction capability is not limited in terms of the number of faults within a configuration frame. External scrubbing can correct any number of faults as long as they do not appear in the configuration bits of the dynamic memory elements; these configuration bits should be masked when performing external scrubbing [112]. In [114], Berg et al. have carried out extensive fault injection analysis to test the performance of a custom external scrubber and a standard Xilinx Virtex-4 internal scrubber. The test results showed that the external scrubber outperformed the internal scrubber in the number of faults correctly detected and repaired.

The goal of configuration memory scrubbing is to avoid the accumulation of soft errors in the system; the efficiency of scrubbing is affected by the number of scrub cycles set by the scrubbing controller. The appropriate scrubbing rate of a particular system will depend on the error rate expected for the system. In [115], Asadi et al. have defined the Mean Time To Manifest (MTTM) term to describe the time a fault

stays inactive in a given system. In an ideal scrubbing scheme, the Mean Time To Detect (MTTD) and the Mean Time To Repair (MTTR) of faults in the system are kept small compared to the MTTM. Setting a high scrubbing rate to enhance the MTTD can increase the power consumption of the system. One approach to address this issue is discussed in [116], wherein the different components of a system are classified according to their criticality; high-priority bits are determined and scrubbed more often than other bits with low priority. Another approach focuses on narrowing the scrubbing region and scrubbing rate by having on-demand scrubbing requests generated by a modular redundancy system [117]. Modular redundancy is one of the most important design concepts in FT systems. TMR is the most common form of redundancy used in FT designs. It is based on triplicating a hardware module to generate three outputs that pass through a voter that performs majority voting to filter out any faulty output of the three. TMR can also be used as a reliable fault detection method whereby comparators are used to determine which redundant module of the three is faulty and trigger a recovery process to repair the faulty module ([117] and [118]). The recovery process can be a scrubbing operation for the affected area or a reconfiguration operation that resets the registers of the faulty module to their initial values. Recovery based on reconfiguration will require the redundant modules to be reset and re-synchronised. This could not be the case with scrubbing as only faults that cause a state change will require a reset after recovery.

TMR is capable of detecting all kinds of faults as long as they are manifested in the output of the affected module. The MTTR in TMR will depend on the size of the triplicated module. Fine-grained TMR designs will have smaller MTTR compared to coarse-grained designs; this, however, comes at the cost of a higher resource overhead due to the additional voting circuitry required [119]. Dual-Modular Redundancy (DMR) can reduce the resource overhead by approximately 1/3 compared to TMR by having only two redundant modules. In DMR, comparators will trigger an error signal in case of a mismatch in the outputs of the two modules (see Figure 3.5). DMR provides the same detection level as TMR; however, in DMR a faulty output is not filtered out, which means that the system must be inactive until the fault is repaired to guarantee correct operation of the system. Moreover, the

MTTR in DMR is increased compared to TMR as there is no mechanism to determine which of the two modules is faulty. So the detect/repair process should be performed in the area covered by the two modules.

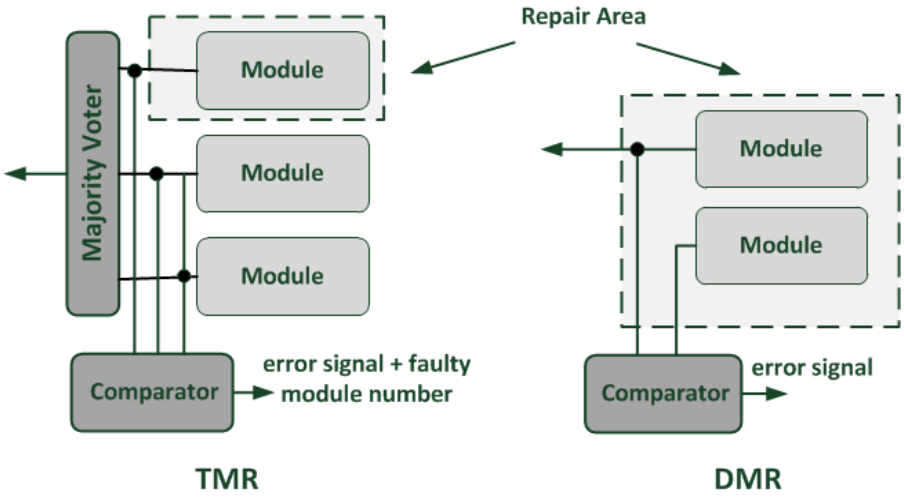



**Figure 3.5 DPR-based fault repair in a redundancy system**

The redundancy concept is not limited to the physical domain of the FPGA; time-domain redundancy for FPGAs was proposed in [120], where an operation is performed twice using the same hardware with different encoding at different times. The results of the two operations are decoded and compared to extract the faulty output. This scheme was shown to have smaller area footprint compared to TMR and DMR at the cost of reduced throughput.

One issue concerning TMR design in FPGAs is the possibility of some faults altering the routing in the design and affecting more than one redundant module causing system failure. There are two approaches discussed in the literature to tackle this problem; the first approach focuses on the floorplan stage of the design. Each redundant module is placed in a distinct region with all of its local routes constrained within the region. These regions are isolated with the appropriate distance of unused resources [121]. The other approach tackles this problem at the RTL design stage by partitioning the design into smaller stages and inserting extra voters to reduce the probability of routing faults affecting more than one redundant module [122].

Another drawback of classic TMR designs is the possibility of single points of failure in the logic of the voting circuitry. As the area of voting circuitry is much smaller than the redundant modules area, the probability of system failure will be reduced to that of the area occupied by the voter. When external scrubbing is used in a TMR system, these voter errors can be corrected at the first scrub cycle. It is also possible to have a triplicated voting path for all the voting stages at the cost of higher resource overhead [123].

**Permanent Faults Mitigation**

Permanent faults are irreversible physical damage in the FPGA resources. The mitigation techniques discussed in the literature are focused on circumventing these resources once they are detected. Similar to soft errors, TMR can detect permanent faults as long as they affect one of the redundant modules outputs. However, TMR can only detect the region affected by a permanent fault and cannot detect the damaged resource within the region. The redundancy system presented in [124] circumvents an entire region occupied by a faulty module in case of a permanent fault and reconfigures the module in a new region to complete the redundancy. This approach can be inefficient because the entire region occupied by the affected module is flagged despite the fact that the damaged resource in the module accounts for a very small portion in the region and this will limit the number of faults that can be mitigated.

Other fault detection methods have been proposed to enhance the granularity of detection. These methods are based on loading different Built-In Self-Test (BIST) circuits offline or online to test the functionality of the FPGA resources. The basic building blocks of a BIST circuit are the Test-Pattern-Generator (TPG), the Circuit-Under-Test (CUT) and the Output-Response-Analyser (ORA). Figure 3.6 demonstrates a basic implementation of a BIST circuit. The TPG generates different data patterns that are passed to the inputs of the CUTs. The CUTs could be as simple as individual LUTs that are configured for a specific function. An ORA is used to compare the outputs of two CUTs; when a faulty CUT is detected, an error signal is generated.

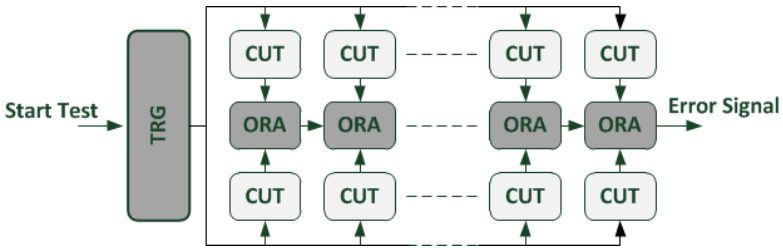



**Figure 3.6 Basic BIST circuit [125]**

The proposed BIST circuits differ in the type of faults they can detect. Some BISTs are focused on the logic blocks ([126] and [127]), while others are focused on interconnect faults ([128] and [129]). Other BIST circuits extend the detection capability to delay faults [130]. BIST circuits can be swapped in and out of the FPGA at run-time using DPR. This technique has been proposed in the Roving Stars fault detection system [131], where fixed-sized test circuits called the 'Horizontal Star' and the 'Vertical Star' are shifted horizontally and vertically to perform a test scan on a given area on the FPGA. These test scans can be performed while other logic outside the scan area remains functional. By dividing the FPGA into equal-sized regions and using some of these regions for the functional blocks in the system, a test block can be swapped between the regions to perform a complete test covering the entire area of the FPGA (Figure 3.7). This method suffers from two main drawbacks: the first is that a minimum of one region has to be empty when floor-planning the design to allow for swapping the functional blocks with the test circuits; the second drawback is the large time overhead of the test operation.

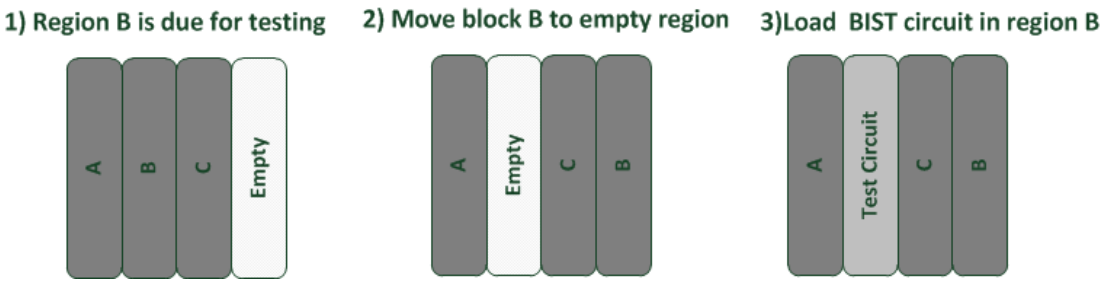



**Figure 3.7 Roving fault detection**

The repair methodologies of permanent faults in FPGAs are based on deactivating the damaged resource and switching its operation to a spare one. Usually the deactivation is performed in segments rather than individual resources. One technique discussed in the literature is the column-based shifting technique presented in [132]. In column-based shifting, the area occupied by a particular design is divided into different columns; the functional blocks of the design are allocated to these columns with some columns left unused by any functional block. Different pre-compiled configurations are then generated, and each configuration has the unused column in a different location. One of these configurations will be the default configuration, when a fault is detected in the default configuration; the fault is mitigated by loading the configuration that has this resource in the unused column.

In [132], Huang et al. also discuss a similar repair technique based on non-overlapping alternate pre-compiled configurations (see Figure 3.8a). In this technique the functional blocks have different arrangements in each configuration but are not constrained to the same area in each configuration. The work based on the multiple pre-compiled configurations was later extended to reduce the storage memory required to store all the pre-compiled configurations by utilising relocatable partial bitstreams ([124] and [133]) (see Figure 3.8b).

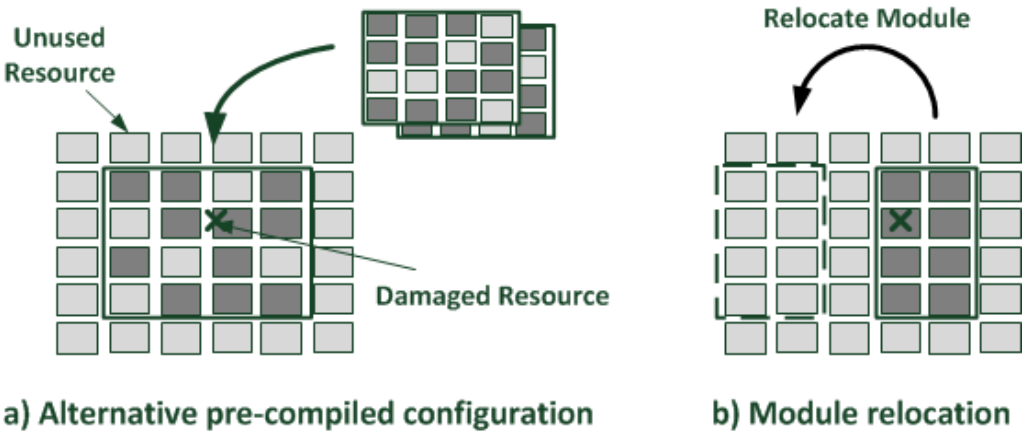



**Figure 3.8 Circumventing damaged resources**

Other methods discussed in the literature take the granularity of repair process to another level in cluster-based FPGAs. The authors in [134] present techniques to work around faults within an FPGA cluster (a cluster is a group of CLBs). If faults within the cluster cannot be avoided, moving to a spare cluster is possible by incremental routing. In [131], Emmert et al. present a system based on a similar concept. In this system faults can be circumvented by loading small configurations called FABRICs into a clustered structure. These FABRICs can either be pre-compiled or computed online. This technique is combined with the horizontal and vertical roving stars, which constantly check for faults in the interconnects.

## 3.3 Chapter Conclusion

This chapter introduced the research work related to the use of DPR for enhanced performance and reliability in FPGAs. This chapter showed the advantages of DPR in hardware/software hybrid systems. Software acceleration can be achieved by off-loading the most performance-demanding portions of the software to hardware accelerators. With DPR, different accelerators can be swapped in/out of the FPGA leading to a more efficient utilisation of the available reconfigurable resources. DPR also opens the door for implementing an ROS where scheduled HTs can be allocated to free areas of the reconfigurable fabric. With the limitations of the configuration throughput in current FPGAs, much of the research work has focused on developing fast and scalable configuration techniques to achieve the throughput required for high-performance systems.

This chapter also looked into the reliability issues preventing the wide-spread use of FPGAs in applications requiring high levels of reliability. The unmatched flexibility of SRAM-FPGAs makes them an excellent solution for space and military applications; however, due to sensitivity of SRAM memory cells to high levels of radiation, FPGAs cannot be deployed in such applications without implementing an efficient fault recovery scheme. This chapter introduced the common fault detection/recovery techniques in FPGAs with special emphasis on the techniques based on the DPR capability in FPGAs. With a combination of design-hardening

techniques and DPR recovery schemes, FPGAs can efficiently handle soft errors in the configuration memory. Moreover, bitstream relocation techniques can also be deployed to mitigate emerging physical defects in the FPGA chip, allowing for greater availability and longer life-time.

# Chapter 4 : A High-Performance Internal Configuration Manager

Efficient internal configuration management is central to self-reconfiguring systems that depend on the ICAP for high-speed dynamic reconfiguration. Typically, internal configuration requires several components implemented in the FPGA to control the loading of partial bitstreams from external memory to the configuration memory of the device. The complexity of the configuration control logic will depend on the requirements of the system. Some systems deploy basic DPR in their operation and only require a simple Finite State Machine (FSM) to control the reconfiguration process. In other systems, such as an ROS kernel implemented on an FPGA, the ICAP is used extensively for different types of operation: task allocation, task de-allocation and writing/reading individual configuration frames. In such systems, the complexity of the configuration control logic is much higher as online modifications to the original partial bitstreams are required to allow for task relocation. There are several design aspects that need to be considered when designing the configuration control circuitry. The design should support various configuration operations in a compact light-weight design. The design should also be highly portable and easily customisable to the needs of a particular system. In addition, the configuration controller should operate at the highest possible throughput to meet the demands of high-performance applications. This chapter presents a novel ICM that addresses all the aforementioned design aspects. The proposed ICM is self-dependent with all the circuitry required to manage the configuration process wrapped in a single top-level module that requires minimal connectivity with the main CPU in the system making it particularly suitable for integration with an ROS kernel. With focus on the Xilinx Virtex family architecture, new methods for enhancing the relocation efficiency and the reconfiguration speed are presented and compared to previous published works.

# 4.1 General Architecture of the ICM

The ICM's architecture is based on separating the low-level configuration particularities from the main CPU in the system. The architecture is tailored for an ROS where a main CPU assigns configuration tasks to the ICM using a software library of configuration functions. These configuration functions can be used for handling the execution of HTs in the available hardware resource in the FPGA as well as handling the inter-task communication through the configuration layer.

## 4.1.1 Building Blocks of the ICM

The ICM consist of three main components: the ICAP controller, the external memory controller and a small soft-processor (see Figure 4.1). The ICAP controller is the core component of the ICM and is responsible for handling the read/write protocols of the ICAP to perform the different configuration operations. To allow for the highest possible throughput when fetching configuration data from the external storage memory, the memory controller has direct access to the external memory module. While typical ICAP controllers depend on the main CPU for initiating the data transfers from external memory to the ICAP over the system's bus [86], the proposed ICM controls all the memory data transfers internally without any assistance from the main CPU. The ICM's soft-processor is a Picoblaze soft-processor, which is optimised for Xilinx FPGAs and has a very small footprint [135]. The ICM's soft-processor is responsible for decoding simple high-level configuration instructions initiated by the main CPU that trigger the execution of certain sub-routines. Each subroutine is intended to trigger and monitor a specific configuration operation such as configuration memory readback, task configuration, task removal, etc. Once the soft-processor decodes a particular instruction, it coordinates the ICM's components according to the requested operation and reports its status back to the main CPU.

### 4.1.2 Interfacing with the Main CPU

The ICM is connected to the main CPU using two 32-bit FIFOs:

**Instruction FIFO**: This FIFO is used by the main CPU to write consecutive configuration requests to the ICM's soft-processor. Small packets are used to request configuration operations. Each packet starts with a request ID number followed by the requested operation command ID and its input parameters. The number of input parameters depends on the type of operation requested by the main CPU. The 'empty' signal of the FIFO is always polled by the ICM's soft-processor to determine if there are configuration operations requested by the main CPU. When packets are sent to the instruction FIFO, the soft-processor starts pulling data out of the FIFO. It first registers the ID number of the instruction and then decodes the operation command. Finally, the soft-processor pulls the operation operands and executes the required sub-routines to perform the operation.

**Status FIFO**: This FIFO is used by the soft-processor to report the ID number of the finished operations. This FIFO is also used to report failed operations as well as sending back output parameters for certain operations.

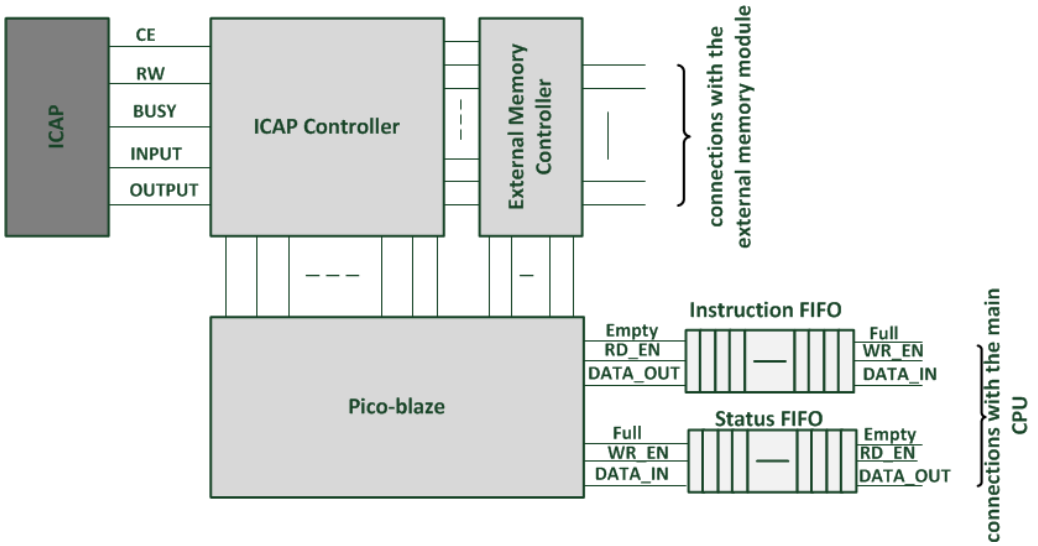



**Figure 4.1 Building blocks of the ICM**

### 4.1.3 The Configuration Operations

The ICM supports different types of configuration operation. These configuration operations can be used in a wide range of applications, including full support for HT management in an ROS. The main configuration operations are described below:

**Partial bitstream configuration**: This operation is intended for reconfigurable modules that are floor-planed according to the Xilinx reconfiguration flow where each reconfigurable module can only be placed in a single reconfigurable region. The partial bitstreams are simply loaded to the ICAP from external memory without any modifications to their content.

**Partial bitstream relocation:** This operation is intended for relocatable modules. The partial bitstreams are modified online according to the chosen locations. This is central to the operation of an ROS as HTs are constantly placed in different locations on the FPGA.

**Black-box configuration**: Typically, when partial bitstreams are generated for reconfigurable modules in the system an extra partial bitstream called the 'black-box' is generated for every reconfigurable region. The black-box basically removes all the logic configured in the region apart from the static routes passing in this region. For relocatable modules, there are several locations on the FPFA where the module can be placed. Each module can have a different shape, making storing an extra black-box for each module impractical and costly in terms of the storage memory. This configuration operation allows for tiling smaller black-box bitstreams horizontally. By initiating several black-box configurations, any region can be 'blanked' provided that the region does not contain any static routes. As each column type contains a different number of minor frames, a black-box bitstream is used for each column type.

**Configuration frames read/write**: There are different situations in which access to individual frames is required. For example, fault injection tests require frames to be read, modified and then written back to the configuration memory. Four operations

are required to perform a fault injection test on a frame. A frame-read operation stores a configuration frame in an internal buffer. A word-fetch operation sends a particular word from the stored frame to the main CPU. A word-write operation replaces a particular word in the internal buffer with a word sent from the main CPU. Finally, a frame-write operation writes the frame stored in the internal buffer to the configuration memory. Table 4.1 summarises the main configuration operations supported by the ICM along with their command IDs and parameters.

**Table 4.1 Main configuration operations**

| Operation | ID | Parameters | Action |
|---|---|---|---|
| ***Frame Read*** | 0x0 | -Frame address<br>-Number of frames | Frames are read from configuration memory and stored in the internal buffer |
| ***Fetch Word*** | 0x1 | -Word number | A word is transferred from the frame buffer to the status FIFO |
| ***Frame Write*** | 0x2 | -Frame address<br>-Number of frames | Frames in the internal buffer are written to the configuration memory |
| ***Write Word*** | 0x3 | -Word number | A word is transferred from instruction FIFO to the frame buffer |
| ***Scrub Frames (ECC)*** | 0x4 | -Frame address<br>-Number of frames | Consecutive frames are read with ECC checking enabled. Only the last frame is stored in the buffer. Automatically corrects corrupted frames. |
| ***Partial Reconfiguration*** | 0x5 | -Partial bitstream ID | Configure a partial bitstream file |
| ***Partial Reconfiguration with Relocation*** | 0x6 | -Partial bitstream ID<br>-Location offsets | Configure partial bitstream in a new location determined by the location offsets |
| ***Black-box Configuration*** | 0x5 | -Number of columns | Configure all ‘0s’ (blank) in a number of adjacent columns |
| ***Clone Partial Bitstream*** | 0x6 | -Partial bitstream ID<br>-Location offsets | Configure a partial bitstream in different locations on the FPGA |

## 4.2 The ICAP Controller

The ICAP controller is the main component in the ICM. The basic building blocks of the ICAP controller are an FSM, an on-chip memory block, a Frame Address Calculator (FAC) and a parallel CRC-32 generator (see Figure 4.2).

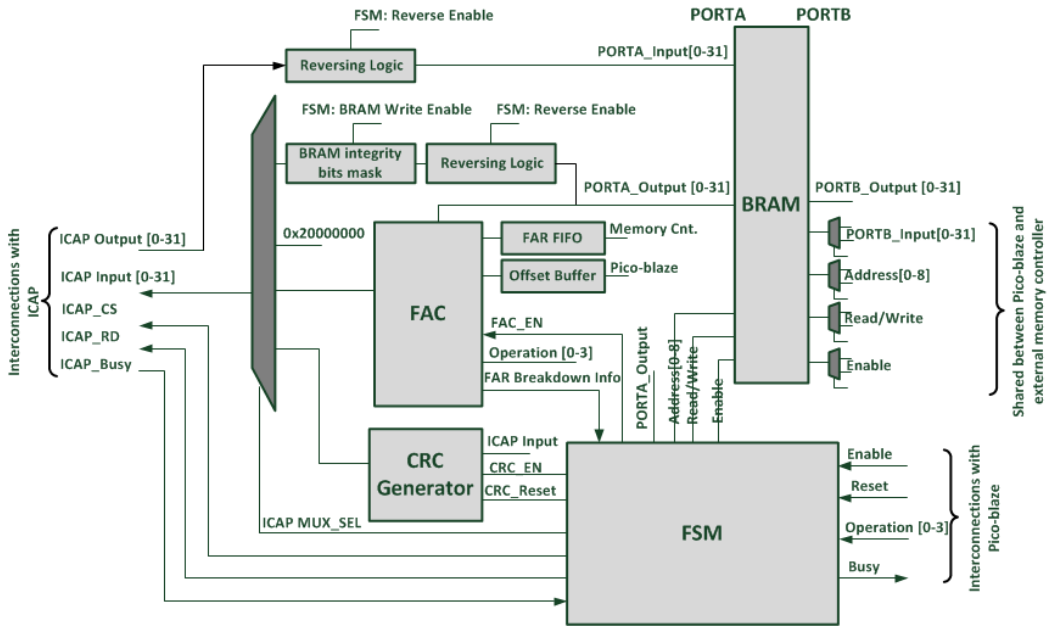



**Figure 4.2 Building blocks of the ICAP controller**

The FSM is responsible for controlling the ICAP signals and the flow of data in/out of the ICAP's input/output ports. Configuration data is transferred between the on-chip memory block and the ICAP ports.

The on-chip memory block is a dual-port BRAM that enables concurrent read and write access using its two ports. The BRAM block is divided into two sections: the first contains several configuration command templates, which are pre-initialised at the RTL-level of the design. The second section is a buffer used for buffering data streamed from the external memory controller as well as for temporary storage of readback data.

To enable fast relocation of partial bitstreams, the algorithm required to modify the partial bitstreams to alter the configuration location is performed in hardware by the FAC. The output of the FAC is multiplexed with the output from the BRAM block and other components in the system that writes to the ICAP. The FSM controls the multiplexer to select which component should access the input port during any configuration operation.

At the end of each partial bitstream, there is a pre-computed CRC value used for configuration verification. During configuration, internal logic in the FPGA computes the CRC value for the configured partial bitstream. When the pre-computed CRC value differs from the value generated by the internal logic, an error flag is set to indicate an error in the configuration process. As relocation involves modifications of the original partial bitstreams, new CRC values must be computed if configuration verification is required. The parallel CRC generator is also connected to the ICAP input multiplexer to alter the CRC value in the input stream when a relocation operation is performed.

The main unique feature of the presented ICM architecture is that all the bitstream modifications required for bitstream relocation are performed using fast hardware component. In many systems, a host processor needs to perform these modifications prior to configuration, which slows down the relocation process (see Chapter 2). In addition, the integration of an internal BRAM, which stores all the configuration command headers allows for fast access to the configuration memory. Systems based on typical ICAP controller, such as the Xilinx Hardware ICAP (HWICAP), do not allow for fast access to the configuration memory as they depend on a host processor to pass the configuration commands from an external memory to the ICAP controller prior to performing any configuration operation. In such systems, passing the configuration commands to the ICAP controller can be performed directly through the system's bus (see Figure 4.3a) or through a Direct Memory Access (DMA) engine to enable burst transfers (see Figure 4.3b).

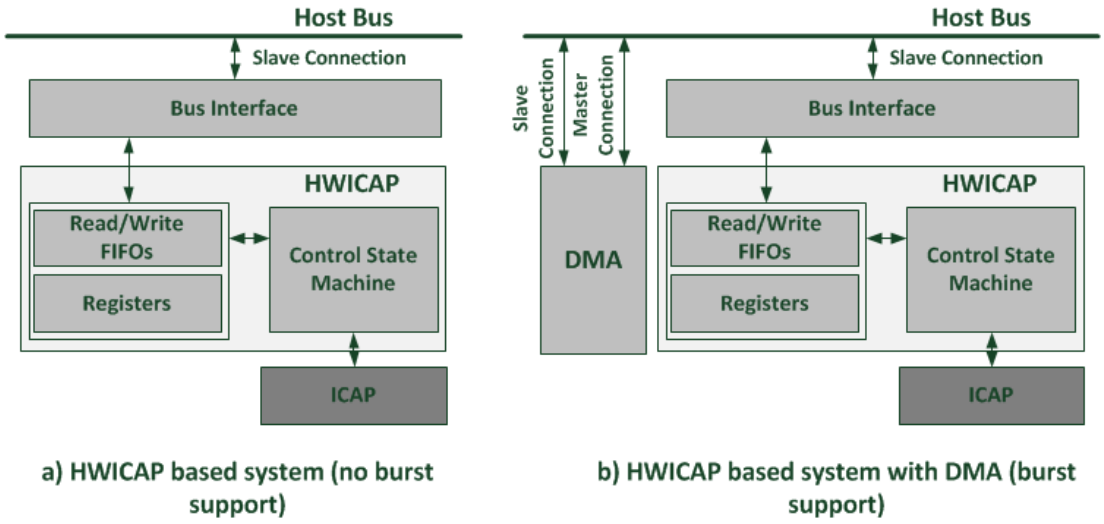



**Figure 4.3 HWICAP based configuration systems [88]**

### 4.2.1 Basic Operation of the Internal Configuration Access Port

The ICAP primitive in Xilinx FPGAs has six connections in its interface: the ICAP clock, the Clock-Enable (CE) signal, the Read/Write (RW) signal, the BUSY signal, the input port and the output port. The CE, RW and BUSY signals are control signals used to control the flow of data in/out of the ICAP. The input and output ports are used to read/write 32-bit words from/to the ICAP. These ports can be configured with different widths: 8-bit, 16-bit and 32-bit.

Read and write operations can be performed using the ICAP to either the configuration memory or the configuration registers. The configuration registers are special registers used to control the operation of the internal configuration logic of the FPGA. Each register has a unique address and can be directly accessed and modified using the ICAP by writing the appropriate command. Xilinx configuration commands have a generic structure, whereby a command is divided into separate fields: Type, opcode, register address and word count (see Table 4.2). The opcode determines if the command is a write, read or a no-operation command by writing '10', '01' or '00' respectively. The register address selects the required register, whereas the word count tells the configuration logic how many words need to be

written/read to/from the selected register. These words immediately follow the command. In some situations, long word sequences are required for reading and writing. When the number of words cannot be set by the fixed size of the 'word count field', two consecutive commands are required to set up the operation. The first command is a type-1 command, which sets the address of the register and the second command is a type-2 command, which has a larger 'word count field' and is used to set the number of words for the operation.

**Table 4.2 Xilinx configuration command structure [25]**

| Header Type | Commands fields bit positions | | |
|---|---|---|---|
| | Opcode | Register Address | Word count |
| 1 | [28:27] | [26:13] | [10:0] |
| 2 | [28:27] | NA | [26:0] |

There are three main configuration registers that control the reading/writing of configuration frames. These configuration registers are the Frame Address Register (FAR), the Frame Data Register-Input (FDRI) and the Frame Data Register-Output (FDRO). The FAR contains the address of the accessed configuration frame. The content written to the configuration frames is written to the FDRI when performing partial reconfiguration or writing to individual frames, whereas configuration memory readback is performed by reading the FDRO register.

The proposed ICM controller goes through three phases to perform an operation: the set-up phase, the data transfer phase and the configuration verification phase. The set-up phase involves preparing the command header for the required operation. This command header contains the ICAP initialisation sequence as well as specific commands for specific registers to control the required operation.

After setting up the required operation, the control enters the data transfer phase where data is transferred to the ICAP. For readback operations, the ICAP must be switched to the read mode during this phase. The switch is performed when a read command is encountered by the controller. The ICAP mode is determined by the RW

signal where logic '0' enables the 'write' mode and logic "1" enables the 'read' mode. The process of switching the ICAP from a read to a write mode or vice versa can be done in three steps: 1) de-assert the CE signal; 2) toggle the RW signal; and 3) assert the CE signal. After setting up a read operation, the readback data will be available in the output port of the ICAP after a number of clock cycles. During this period the BUSY signal of the ICAP remains high, indicating that the readback data is not available yet. When performing a read operation of a configuration frame, the required configuration data will appear in the output port of the ICAP after a dummy frame and a dummy word. The same applies for writing configuration frames during the data transfer phase. After writing the last configuration frame, an extra dummy frame must be written to the ICAP, however, no dummy word is required for write operations.

The final phase is when the configuration verification is performed and the ICAP is desynchronised to return to the idle state. This operation is performed by sending special configuration command trailers that are required to de-synchronise the operation. In case of a read operation, the ICAP must be switched back to the write mode before sending these commands.

### 4.2.2 Fast Operation Set-up

In the proposed ICAP controller, pre-generated command header and trailer templates are stored in the dual-port BRAM. Each type of operation has dedicated command templates. The command templates are stored in the top half of the BRAM, whereas the remaining empty memory locations are used as a read/write buffer (see Figure 4.4).

Each template contains some fields that represent the variable parameters specific for the required operation such as the frame address and the number of configuration frames to read/write. These fields are accessible by the ICM's soft-processor, and can be initialised with the required values before initiating the required operation. The BRAM is dual port; one port is dedicated for the controller to access the operation templates and the read/write buffer and the other port is shared between the

soft-processor and the external memory controller through multiplexers (see Figure 4.2). When the soft-processor receives a particular instruction from the main CPU in the systems, it modifies the fields in the required templates before triggering the operation. The command templates are divided into three groups: the configuration memory write, the configuration memory read and the MFW templates.

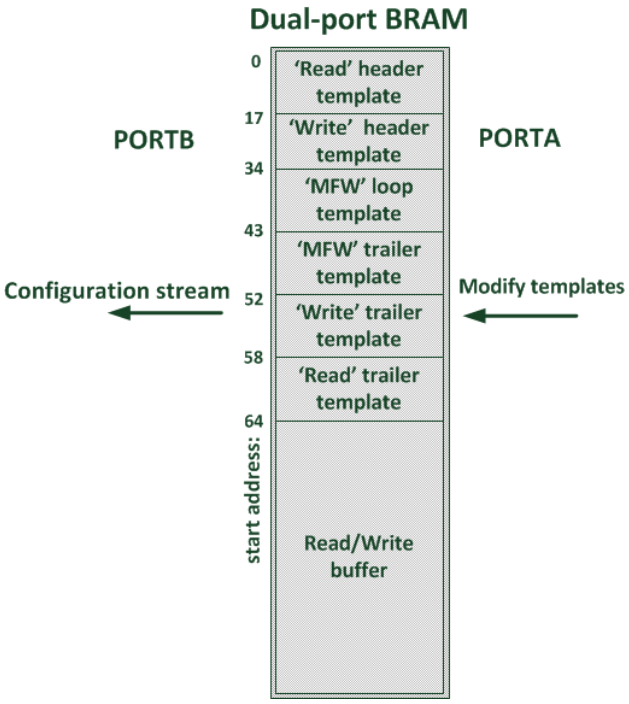



**Figure 4.4 The dual-port BRAM block**

### Configuration Memory Write Templates

There are two templates stored in the dual-port BRAM for the configuration memory write operation. The main fields that can be modified in the header template are: the frame address, the number of words to be written, the ID code of the device, the Control Register (CTL) and the MASK registers values.

The FAR value specifies the address of the first frame to be written to the configuration memory. The FAR address only needs to be set once, as it is automatically incremented for consecutive frames in the FPGA's configuration circuitry. To set the number of frames to be written consecutively, the word count field of the FDRI-write command is modified.

Each Xilinx FPGA model has a unique ID that needs to be set in the ID register before a write operation. The CTL and MASK registers are used to enable/disable writing to the LUTs configured as shift registers or distributed RAM by writing logic '1'/'0' to the GLUTMASK bit in the CTL register. Writing to the CTL register is masked by the MASK register, so two fields need to be modified to change the GLUTMASK bit in the CTL register. Table 4.3 shows the commands templates used for the configuration memory write operations.

**Table 4.3 Writing command templates**

| *Template Type* | *Configuration Data* | *Explanation* |
|---|---|---|
| ***Header Template*** | AA995566 | Synchronisation word |
| | 20000000 | NO-Operation command |
| | 30008001 | Write 1 word to command register |
| | 00000007 | Reset CRC command |
| | 20000000 | NO-Operation command |
| | 20000000 | NO-Operation command |
| | 30018001 | Write 1 word to ID register |
| | xxxxxxxx | Device ID |
| | 3000C001 | Write 1 word to MASK register |
| | xxxxxxxx | Mask value |
| | 3000A001 | Write 1 word to CTL register |
| | xxxxxxxx | CTL register value |
| | 30002001 | Write 1 word to the FAR register |
| | xxxxxxxx | The frame address value |
| | 30008001 | Write 1 word to the command register |
| | 00000001 | WRITE configuration data command |
| | 30004xxx | Write (xxx) words to the FDRI register |
| **The configuration stream must be switched to the read/write buffer**<br>**Dummy word + frames + dummy frame = (xxx)** | | |
| ***Trailer Template*** | 30000001 | Write 1 word to CRC register |
| | xxxxxxxx | CRC-generator inserts checksum here |
| | 30008001 | Write 1 word to command register |
| | 0000000D | Write de-synchronisation command |
| | 20000000 | No-Operation command |
| | 20000000 | No-Operation command |

### Configuration Memory Read Templates

Similar to the configuration memory write operation, the read operation has two command templates stored in the BRAM. The variable fields in the read header template are: the CTL register field, the MASK register field, the word count of the FDRO-read command and the FAR field. Table 4.4 shows the command templates for the configuration memory read operation.

**Table 4.4 Reading command templates**

| *Template Type* | *Configuration Data* | *Explanation* |
|---|---|---|
| ***Header Template*** | AA995566 | Synchronisation word |
| | 20000000 | NO-Operation command |
| | 30008001 | Write 1 word to command register |
| | 00000007 | Reset CRC command |
| | 20000000 | NO-Operation command |
| | 20000000 | NO-Operation command |
| | 30008001 | Write 1 word to command register |
| | 00000004 | READ configuration data command |
| | 3000C001 | Write 1 word to MASK register |
| | xxxxxxxx | Mask value |
| | 3000A001 | Write 1 word to CTL register |
| | xxxxxxxx | CTL register value |
| | 30002001 | Write 1 word to the FAR register |
| | xxxxxxxx | The frame address value |
| | 28006xxx | Read (xxx) words from the FDRO register |
| | 20000000 | NO-Operation command |
| | 20000000 | NO-Operation command |
| **Data from the ICAP output port must be stored in the read/write buffer**<br>**Dummy word + dummy frame + frames = (xxx)** | | |
| ***Trailer Template*** | 20000000 | NO-Operation command |
| | 20000000 | NO-Operation command |
| | 30008001 | Write 1 word to command register |
| | 0000000D | Write de-synchronisation command |
| | 20000000 | No-Operation command |
| | 20000000 | No-Operation command |

### The Multiple Frame Write (MFW) Command Templates

The MFW feature in Xilinx FPGAs allows for writing the same frame to several addresses in a single operation rather than writing each frame individually. This operation is commonly used for offline bitstream compression, where the BitGen tool looks for configuration frames with similar content and uses the MFW commands to enable the storing of their content just once instead of multiple times in the bitstream file. The MFW requires the frame to be written first to the FDRI register; the frame then can be copied to a new address by using a special set of commands. This set of commands can be repeated as desired to copy the written frame to multiple addresses in a much shorter configuration time. Figure 4.5 shows the command sequence of the MFR operation compared to the normal write operation.

Since the number of times the MFW command sequence is required in a single operation will depend on the number of frames to be copied, a 'loop' template is added to the command templates, which can be used in several iterations during the configuration process. The MFW header template is the same as the configuration write template with the number of words in the FDRI-write command fixed for a single frame as no dummy frame is required for the MFW configuration. Table 4.5 shows the command templates for the MFW configuration.

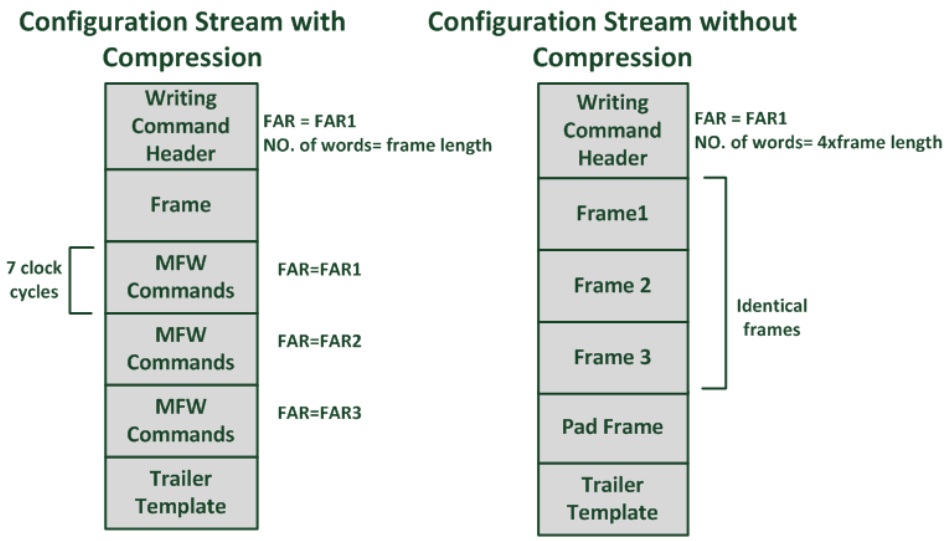



**Figure 4.5 Writing three identical consecutive frames with and without compression**

**Table 4.5 MFW command templates**

| *Template Type* | *Configuration Data* | *Explanation* |
|---|---|---|
| ***FAR*** | 30002001 | Write 1 word to FAR |
| | xxxxxxxx | FAC inserts FAR here |
| ***Loop Template*** | 30008001 | Write 1 word to command register |
| | 00000002 | Write the MFW command |
| | 30014002 | Write 2 words to the MFW register |
| | 00000000 | Dummy word |
| | 00000000 | Dummy word |
| | 30002001 | Write 1 word to FAR |
| | xxxxxxxx | FAC inserts FAR here |
| ***Trailer Template*** | 30014002 | Write 2 words to the MFW register |
| | 00000000 | Dummy word |
| | 00000000 | Dummy word |
| | 30000001 | Write 1 word to CRC register |
| | xxxxxxxx | CRC-generator inserts checksum here |
| | 30008001 | Write 1 word to the command register |
| | 0000000D | Write de-synchronisation command |
| | 20000000 | NO-Operation command |
| | 20000000 | NO-Operation command |

### 4.2.3 The Data-Transfer Phase

After the soft-processor finishes modifying the command templates according to the instruction sent by the main CPU, it triggers the ICAP control's FSM, which is responsible for coordinating the controller's components to perform the required operation. It is noted that some of the variable fields are not modified in the setup phase such as the CRC and the FAR field in the MFW loop template. These fields are modified during the streaming of the configuration data by the CRC-generator and the FAC.

One port of the dual-port BRAM is dedicated for the configuration data stream. The port contains a 32-bit input and a 32-bit output. The port is controlled by three signals: the data address, the enable signal and the read/write signal. The FSM

controls these signal to generate the configuration data for the required operation. The on-chip BRAMs are fast; they support high operational frequency and a very low latency (one clock cycle). With the appropriate address control, any command sequence can be generated on the output port of the BRAM.

The three main states in the data transfer phase are the Command-Write, the Data-Write and the Data-read states. The Command-Write state is executed while passing configuration commands to the ICAP. In this state, each configuration command is decoded to extract some parameters and to set some flags required to control the data-transfer phase.

Two parameters are extracted from each command: the command type and the word count. The command type and word count are determined according to Table 4.2, whereby the command could be registered as either a write command or a read command. The word count contains the number of configuration words to be written or read after the decoded command. The next state is determined according to the command type and number of words, as shown in Figure 4.6.

When a write command to the FAR is encountered in the Write-Command state, a FAR flag is set to indicate that the next word is a frame address. This flag is monitored by the FAC, which performs modifications to the frame addresses for some operations. In addition, the FAC breaks down the FAR to extract the resource type as well as the original frame location from the FAR.

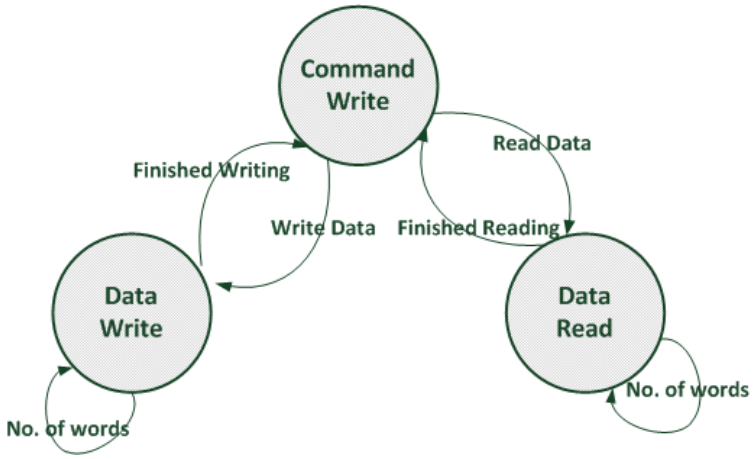



**Figure 4.6 Main states in the data transfer phase**

### Configuration Memory Read/Write

The basic configuration memory write/read operations do not require further modifications in the command templates other than the modifications carried out by the soft-processor in the setup phase.

The same applies for writing the configuration data stored in the read/write buffer unless the resource type registered by the FAC is indicating a BRAM resource type. The content of the BRAM configuration data contains several protection bits all set at logic '1'. In Virtex-4 FPGAs, these bits are placed on the 24$^{th}$ bit of the of the 4$^{th}$, 14$^{th}$, 25$^{th}$ and 35$^{th}$ minor frames and the 8$^{th}$ bit of the 5$^{th}$, 15$^{th}$, 26$^{th}$ and 36$^{th}$ minor frames of a BRAM column. These bits need to be cleared to enable writing to the BRAM column. When the FAC detects a BRAM resource type, dedicated combinatorial logic clears these bits before passing the configuration frames to the ICAP.

The Virtex FPGA architecture consists of several rows of resource columns. These rows are divided between the two halves of the FPGA. The configuration frames of columns in the bottom half of the FPGA have a reversed bit order compared to the frames of the columns in the top half of the FPGA, with the exception of the word containing the ECC, which is the 21$^{st}$ word in Virtex-4 FPGAs. This means that if a configuration frame from the top half of the FPGA is required to be copied to a location in the bottom half of the FPGA or vice versa, the order of the frame words must be reversed as well as the order of bits in each word.

To enable the read-relocate-write feature between the top and bottom halves of the FPGA, a frame bit reversal is always performed when writing configuration data to the bottom half of the FPGA by performing an address jump to the last word of the frame in the read/write buffer and decrementing the address until the first word is reached. The order of bits within each word is reversed by dedicated combinatorial logic before passing the data to the ICAP. For read operations, the same is done when storing configuration frames read from the bottom half of the FPGA. Figure 4.7 shows the different scenarios for configuration memory read and write operations.

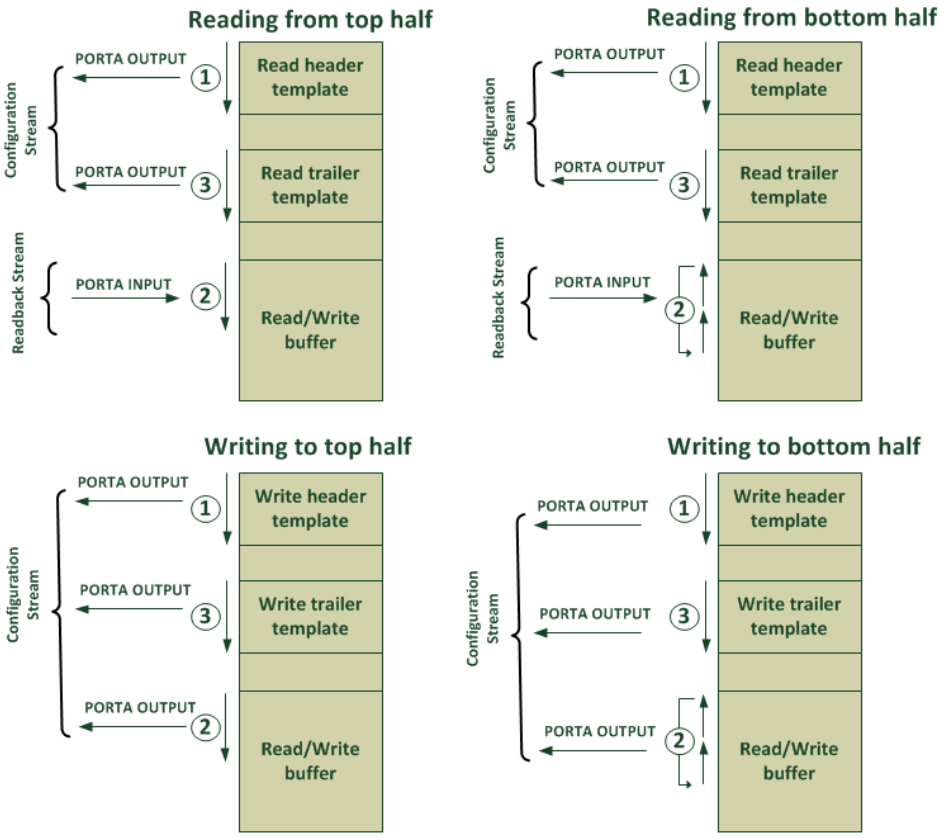



**Figure 4.7 Transfer phase for read/write operations**

### Partial Reconfiguration

The basic partial reconfiguration operation supported by the ICAP controller allows for a partial bitstream file to be to be passed to the ICAP input port after initialisation without any modifications to its content. No templates are used for this operation as all the commands required are inside the bitstream file. For this operation, the FSM controls the transfer of data from the external memory controller to the ICAP through the read/write buffer in the dual-port BRAM. In the setup phase, the soft-processor grants access to the shared port of the BRAM to the external memory controller by setting the 'select' signals of the port multiplexers. The memory controller writes the configuration data in the buffer starting from the first address of the buffer region in

the BRAM. When the external memory reaches the end of the buffer, a jump to the buffer's first address is performed. This procedure is repeated until all the bitstream is written. Reading the configuration data from the other port, which is controlled by the FSM, is synchronised with the port accessed by the external memory controller (see Figure 4.8). It is noted that the soft-processor enables the external memory before the FSM to allow for the latency of the external memory module.

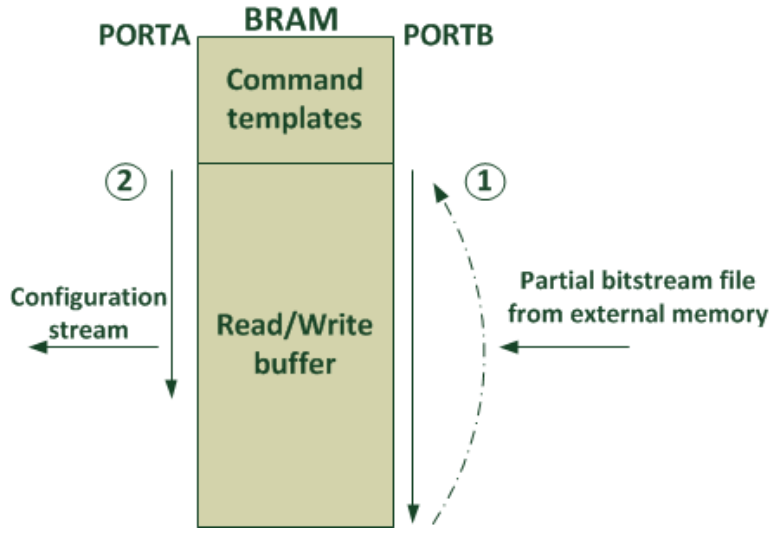



**Figure 4.8 Transfer phase for basic partial reconfiguration**

### Offset-based Bitstream Relocation

Bitstream relocation requires modifying the FAR values in the bitstream to change the location of configuration in the FPGA. All Xilinx Virtex FPGAs have similar frame addressing architecture. A configuration frame is the smallest addressable segment of data. There are different types of configuration frame for different types of logic resource (IOB, CLB, DSP, clock resources and BRAM). Each frame is configured in the location indicated by the FAR. The FAR is divided into five fields: top/bottom, block type, row address, column address and minor address. Figure 4.9 shows a generic physical layout of a Xilinx Virtex FPGA device.

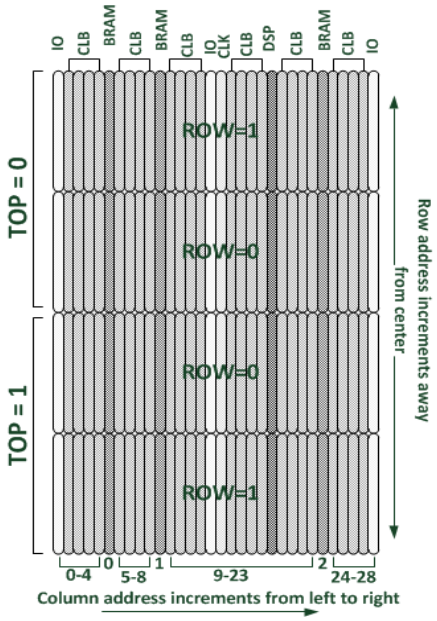



**Figure 4.9 Frame addressing in Xilinx Virtex FPGAs**

The FPGA is divided into two halves. The top half is addressed by '0' and the bottom half is addressed by '1' in the top/bottom field of the FAR. Each half is divided into two rows. In each half of the FPGA, the row address starts with '0' near the centre of the device and increments moving away from the centre of the device. The row is also divided into vertical columns of different types of FPGA resource. The type of column is specified by the block type field in the FAR ('00' for IOB, CLB, CLK and DSP; '01' for block RAM interconnects; '10' for block RAM content). A configuration frame configures resources spanning the entire height of a column. This means that the smallest partial bitstream contains the configuration data for a single column. Each column consists of a fixed number of minor frames depending on the type of column. Xilinx FPGAs support an auto-increment feature of the FAR value for the configuration of consecutive frames of the same resource type. This means that an uncompressed partial bitstream of a reconfigurable module consisting of only horizontally adjacent columns of the same type contains a single write to the FAR. Modifying the original FAR value for relocation in such bitstreams is simple as only a single modification is required. Unfortunately, this is not the case for most partial bitstreams. Compressed partial bitstreams utilise the MFW command

sequence, which requires several FAR writes. The number of FAR writes in a compressed partial bitstream will depend on the number of compressed frames (see Figure 4.5). In addition, the number of FAR values in uncompressed partial bitstreams depends on the resource types within the RM and the column arrangement of the RM. These FAR values must be modified according to the target location of the bitstream.

The proposed relocation method in this thesis aims at eliminating any time overhead incurred from the FAR modifications by performing the modifications using dedicated logic while partial bitstreams are configured. The FAC in the ICAP controller is responsible for calculating each new FAR value required for relocation by extracting some information from the original FAR values and manipulating them with location offsets passed by the soft-processor. The soft-processor sends a horizontal offset 'X' and a vertical offset 'Y' to the FAC to indicate the target location of configuration. The X offset is divided into two fields: '$X_{clb}$' and '$X_{BRAM}$'; this is because BRAM columns has a different column addressing index compared to the other resources as shown in Figure 4.9

The proposed relocation model assumes that all the partial bitstream files are generated for the location as close as possible to the top-left corner of the FPGA. With this assumption, shifting the modules horizontally becomes a matter of adding a horizontal offset 'X' to the column address extracted from the FAR.

Shifting reconfigurable modules vertically is more complex as the row address increments away from the centre of the FPGA. A 'Y' offset is used according to Algorithm 4.1 to find the required row address.

***IF*** $Y_{offset}$ < *Number of rows in one of the FPGA's halves*
    $Row\ Adress_{relocation} = Row\ Address_{original} - Y_{offset}$
***Else***
    $Row\ Adress_{relocation} = Y_{offset} - Row\ Address_{original} - 1$
***END IF***

**Algorithm 4.1 Calculating relocation row address from Y offsets**

These FAR modifications are performed when the FAC is enabled by a 'FAR flag' that is set in the Command-Write state. The output of the FAC is passed to the ICAP by switching access to the ICAP input port to the FAC for one clock cycle.

As mentioned earlier, the configuration frames of the columns located in the bottom half of the FPGA have a reversed bit order compared to columns located in the top half of the FPGA. This will limit the possible locations of the bitstream to a single half of the FPGA. Configuration frames bit reversal is performed for any module configured in the bottom half of the FPGA to tackle this problem (see Figure 4.10). When configuring with frame bit reversal, writing to the read/write buffer by the external memory controller is enabled well ahead of the other BRAM port to allow for the required word reversal (41 clock cycles for Virtex-4).

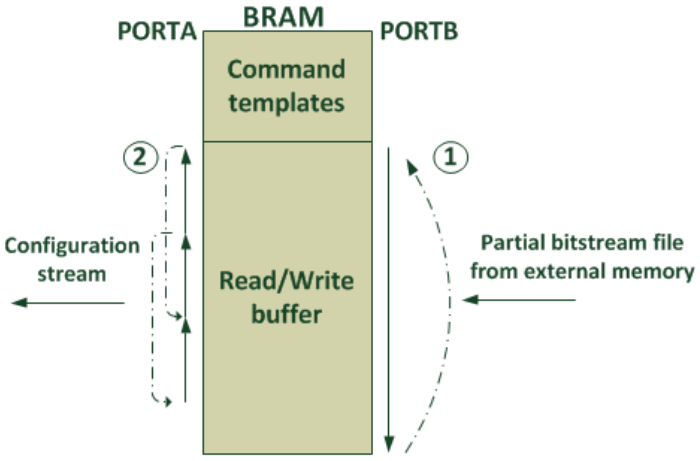



**Figure 4.10 Transfer phase when relocating to the bottom half of the FPGA**

### Black-box Configuration

Removing already configured modules that are inactive is essential to reduce the overall power consumption of the system. Each relocatable module could have a different size. It may be inconvenient to store a black-box configuration for each relocatable module. Since relocatable modules are mostly configured in locations with no static routes, their removal process is a matter of writing empty frames in the

entire region they occupy. Empty frames are frames with all bits equal to '0'. Normal write operations can take a long time to 'blank' a given area, especially when each frame is written individually. This is because each write operation will require an extra dummy frame to be written to the FDRI register. Writing a larger number of frames in a single operation will reduce the overall configuration time. However, black-box bitstreams are usually compressed as they contain a large number of empty frames that can be written once using the MFW feature.

The proposed solution for blanking any arbitrary area on the FPGA is to generate the required compression commands online using the MFW command templates stored in the dual-port BRAM. To easily blank an area of several resource columns, each column is assigned with a separate MFW configuration operation where the loop MFW command template is used several times to blank all the minor frames inside the column. The blanking process of a particular column starts by writing an empty frame to the FDRI register. To write the empty frame, the address of the BRAM port that generates the configuration stream is stalled at a reserved empty memory location for the required number of clock cycles. After writing the empty frame, the compression commands are generated by passing the loop MFW command template several times. In each iteration of the MFW command template, the FAR value is updated by the FAC which increments the minor frame field. Figure 4.11 shows the blanking operation.

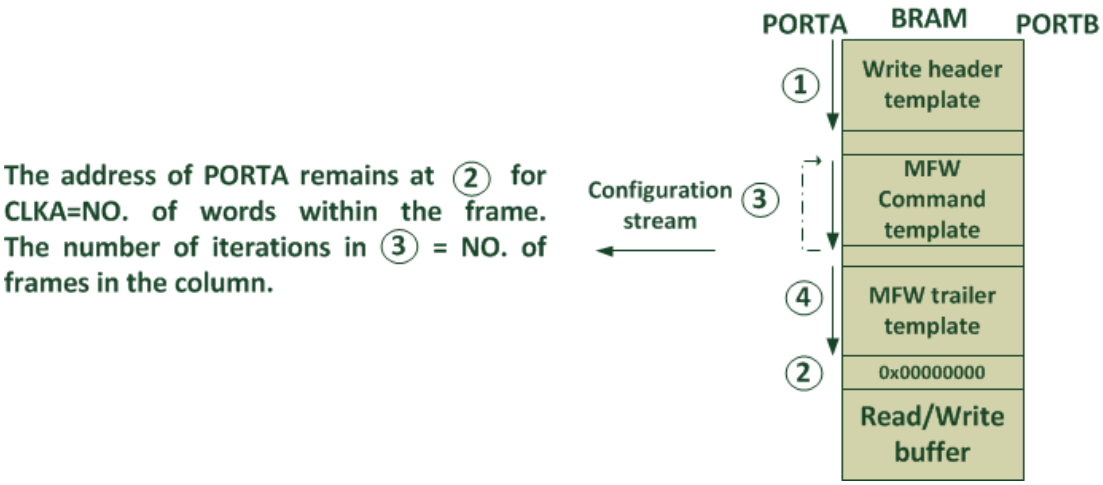



**Figure 4.11 Transfer phase for black-box configuration**

### 4.2.4 The Configuration Verification Phase

During configuration, the internal configuration logic of the FPGA calculates a CRC value for the configured data. The CRC calculation starts after a 'reset CRC' command is written to the command register. When generating a partial bitstream, the BitGen software tool generates a CRC value that is written to the CRC register at the end of the partial bitstream file. This value is compared to the value calculated by the internal logic of the FPGA and an error flag is set in the "Status" register in case of a mismatch.

As relocation involves modifying the original partial bitstreams of the relocatable modules, the original CRC values becomes invalid. As all the modifications to the bitstream are performed during the configuration process, the new CRC value must be calculated using a parallel CRC-generator, which processes a 32-bit word every clock cycle. Xilinx FPGAs use a standard CRC-32C checksum algorithm defined by the following polynomial:

$$X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^{9} + X^{8} + X^{6} + 1$$

There are several methods for creating parallel CRC generators. Xilinx provides a parallel CRC generator tool that generates an HDL code for the parallel 32-bit CRC-generator [136]. The CRC-generator output is connected to the ICAP input multiplexer. When a write to the CRC register is detected in the Write-Command state, the output of the CRC generator is selected.

It is noted that the implementation of the CRC generator is optional for configuration through the ICAP controller. Writing a wrong CRC value to the CRC register does not cause any corruption in the configuration process.

## 4.3 The External Memory Controller

As explained earlier, the external memory controller is responsible for moving partial bitstreams from the external memory to the read/write FIFO of the ICAP controller. To move a particular partial bitstream, the external memory needs to know the

address of the segment of memory storing this partial bitstream as well as the size of this segment. To ease the management of a large number of partial bitstreams, a dedicated segment in memory called the pointer table is used to store the addresses of the first memory location of each partial bitstream file stored in memory. Each partial bitstream is assigned with an ID number, this ID number represents the order of the partial bitstream in the pointer table. The external memory controller only stores the start address of the pointer table and the start/end addresses of the buffer in the ICAP controller. To configure a particular partial bitstream, the ID number of this partial bitstream is passed to the memory controller. The external memory controller reads the address of the first memory location of the file from the pointer table and then reads the partial bitstreams file header, which contains the size of the file (see Figure 4.12). The memory controller writes directly to the buffer in the ICAP controller according to the scheme shown in Figure 4.8. The ICAP controller's FSM is responsible for synchronising the data transfer in and out of the buffer. To do this, the FSM monitors the last address in the buffer accessed by the memory controller.

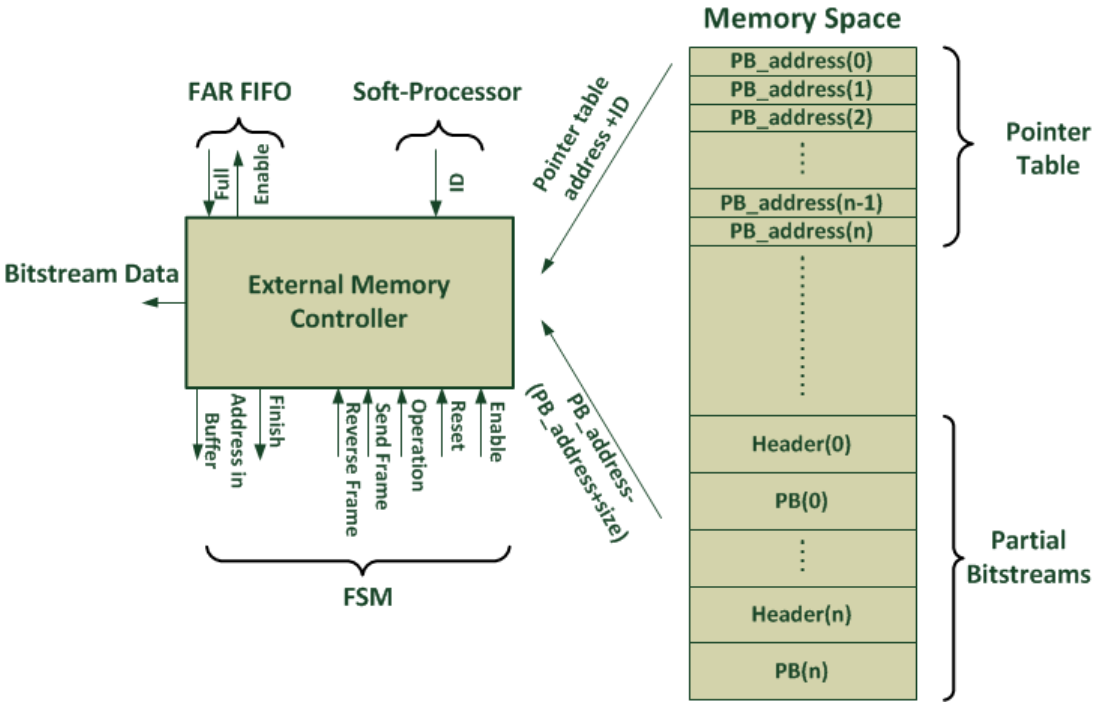



**Figure 4.12 The external memory controller**

Two versions of the external memory controller were designed to support two popular types of memory module: Zero Bus Turnaround (ZBT) SRAM and Double Data Rate (DDR) SDRAM.

The ZBT SRAM memory is very popular for applications requiring low latency. There is only two clock cycles latency when reading a specific address; this is very efficient for non-burst data transfers (e.g. jumping between segments in memory). In addition, the ZBT SRAM interface is very simple requiring only a small controller implemented in the FPGA.

DDR SDRAM is widely used when a larger size of memory is required in the system or when several components in the system need direct access to the memory. Xilinx provides an optimised memory controller for its FPGAs. The MPMC IP core from Xilinx provides support for DDR, DDR2, DDR3 SDRAM memory modules [137]. The MPMC contains eight ports, which provide access to memory through one of the standard Personality Interface Module (PIMs). The fastest PIM standard for the MPMC port is the NPI, which provides low latency direct access to the memory. The MPMC supports burst transfer lengths of up to 64 words. However, the data latency is large compared to the ZBT SRAM. The designed NPI-MPMC controller achieved a latency of 30 clock cycles between burst data transfers.

The partial bitstreams are copied from a non-volatile memory to the RAM memory module by the main CPU after power-up. Multiplexers are used to switch access to the ZBT SRAM memory module between the main CPU and the ICM, whereas a dedicated port of the MPMC memory controller is used to connect the main CPU. It is noted that the ICM is optimised for the SRAM controller and the rest of this chapter assumes the SRAM controller as the default controller.

## 4.4 Multiple-Clone Configuration

Due to the sequential nature of the configuration logic in FPGA, the configuration process is limited to the maximum theoretical throughput of the configuration port. Most of the attempts reported in the literature to reduce the configuration time

overhead are based on over-clocking the configuration port or optimising the reconfigurable modules to realise smaller partial bitstreams (see Chapter 3).

Parallel reconfiguration is unfeasible in current commercial FPGAs. Parallel reconfiguration can be defined as the ability to reconfigure several regions on the FPGA at the same time. If we examine the MFW configuration supported in the latest Xilinx FPGAs, we can see that it allows for some configuration parallelism but with some limitations. The MFW allows for writing several frames at once, provided that these frames have the same content. In other words, it clones a configuration frame in several locations on the FPGA. Typically, the MFW is utilised offline to reduce the number of frame repetitions in the bitstream file and achieve compression which depends on the original design. It was shown in earlier sections of this chapter how the MFW commands can be managed online to realise an efficient method for black-box configuration. This section explains how the online configuration management can be extended to allow for parallel configuration of identical relocatable modules (i.e. clones) using a proposed multiple-clone configuration technique. This technique allows for online generation of a single bitstream that configures different instances of the same module in different locations on the FPGA in much shorter configuration time compared to normal configuration.

The multiple-clone configuration feature is fully integrated with the presented ICAP controller. Figure 4.13 highlights three applications that can benefit from such configuration techniques.

The first application is the configuration of redundant modules in an N-Modular Redundancy system. In such systems an ‘N’ number of redundant modules is used to perform the same process to enhance the reliability of the system. A redundancy system can be designed as a partially reconfigurable system where the redundant modules can be swapped to perform different functions [117].

The second application is an embedded systolic array acceleration system, which contains several configuration slots for the configuration of hardware systolic array accelerators ([69] and [70]). Several slots can be connected together to form a larger systolic array depending on the level of acceleration required.

This configuration technique can also be deployed in an ROS where several tasks are executed in hardware. When successive tasks that use the same hardware are scheduled for execution, this technique can be used to reduce the configuration time and consequently ease the load on the configuration port.

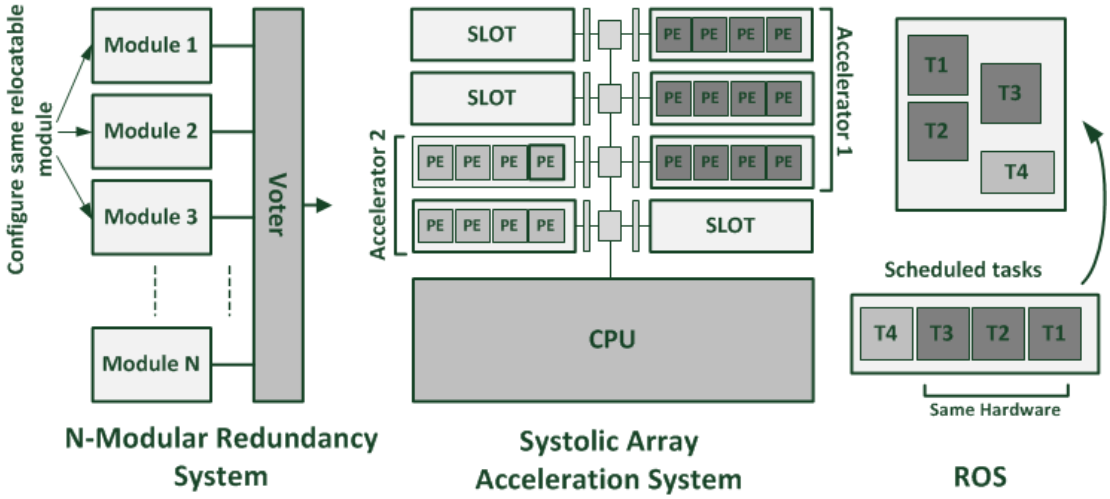



**Figure 4.13 Possible applications of the multiple-clone configuration**

### 4.4.1 Overview

To clone a particular module in several locations on the FPGA, each location is assigned with an 'X'/'Y' offset pair. Rather than writing the configuration frames sequentially to the FDRI, each frame is written separately to the FDRI followed by several iterations of the MFW commands. For non-compressed partial bitstreams, the number of MFW command iterations for each frame is equal to the number of clones required. In each iteration, the FAR value is recalculated for a new offset pair. After finishing all the MFW command iterations of a frame already written to the FDRI, the next frame is written to the FDRI and the same process is repeated until all the clones are configured (see Figure 4.14).

For compressed partial bitstreams, the number of iterations of the MFW commands can vary between frames as some frames already deploy these commands in the original partial bitstream for compression. For each compressed frame, the number of MFW command iterations is equal to (number of clones x number of compressions).

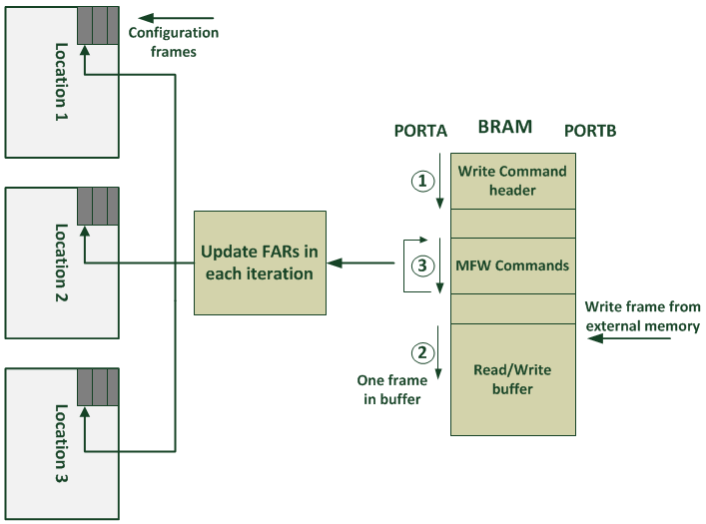



**Figure 4.14 Multiple-clone configuration**

## 4.4.2 The Clonable Partial Bitstream

Partial bitstreams generated by the Xilinx software tool do not contain the FAR value of each frame in the bitstream. The internal logic of the FPGA automatically increments the FAR when adjacent frames are written to the FDRI. For the multiple-clone configuration, the FAR of each frame must be pre-computed to allow for fast update of the FAR field in the MFW command template.

A simple C code was developed to decode the Xilinx Virtex partial bitstream and calculate the addresses of individual frames. This code is used to create a new bitstream file with all the information necessary for the cloning process. The structure of the generated bit file is designed to be easily accessible by the ZBT SRAM memory controller to perform both normal configuration as well as multiple-clone configuration.

The bitstream file consists of a file header and several frame segments, one for each frame in the bitstream (see Figure 4.15). These frame segments are grouped according to the resource type they configure. The CLB frame segments are placed first in the file followed by the BRAM, BRAM interconnects and DSP frame segments. The file header basically contains the number of frame segments in the file. Each frame segment contains its own header. This header contains a compression field indicating the number of compressions for the frame in the original bitstream. If no compression operation is associated with the frame, the frame segment header is '0'. Any frame is always preceded by its pre-computed FAR value. The compression addresses always follow the frame in case it was used in a compression operation.

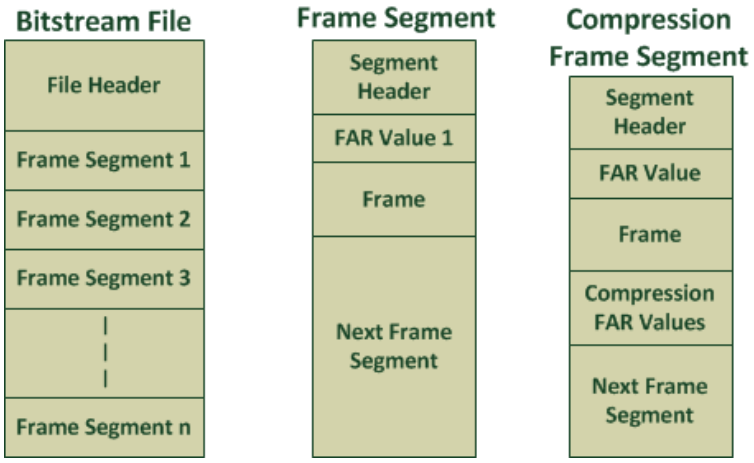



**Figure 4.15 The clonable bit file**

## 4.4.3 The Configuration Process

All the information required to perform the cloning process is contained in the clonable bit file. To clone a module in several locations on the FPGA, the ICAP controller's soft-processor passes an offset pair for each location to a buffer connected to the FAC. During configuration, the frame segments are processed separately. The external memory controller passes each frame to the read/write buffer in the ICAP controller and any FAR value associated with the frame to a FAR

FIFO connected to the FAC (see Figure 4.16). The FSM synchronises the frame fetching from the external memory so that it does not cause any interruption in the configuration stream. This is possible due to the very low latency of the ZBT SRAM. A 'Send Frame' signal connected to the memory controller is pulsed every time a frame transfer is required. Writing to the FDRI can start once the first word of the frame is available in the buffer. In case the cloning configuration process is performed in the bottom half of the FPGA, a 'Reverse Frame' signal is triggered to instruct the memory controller to fetch the word in reverse order.

When writing the MFW commands, the FAC calculates the FAR values for each clone according to the values present in the FAR FIFO and the offset buffer. Each FAR value is utilised in several MFW command iterations until it is used with all the offset pairs in the buffer. The FSM will continue looping through the MFW command template until the FAR FIFO is empty. After that, cloning proceeds to the next frame segment in the bitstream.

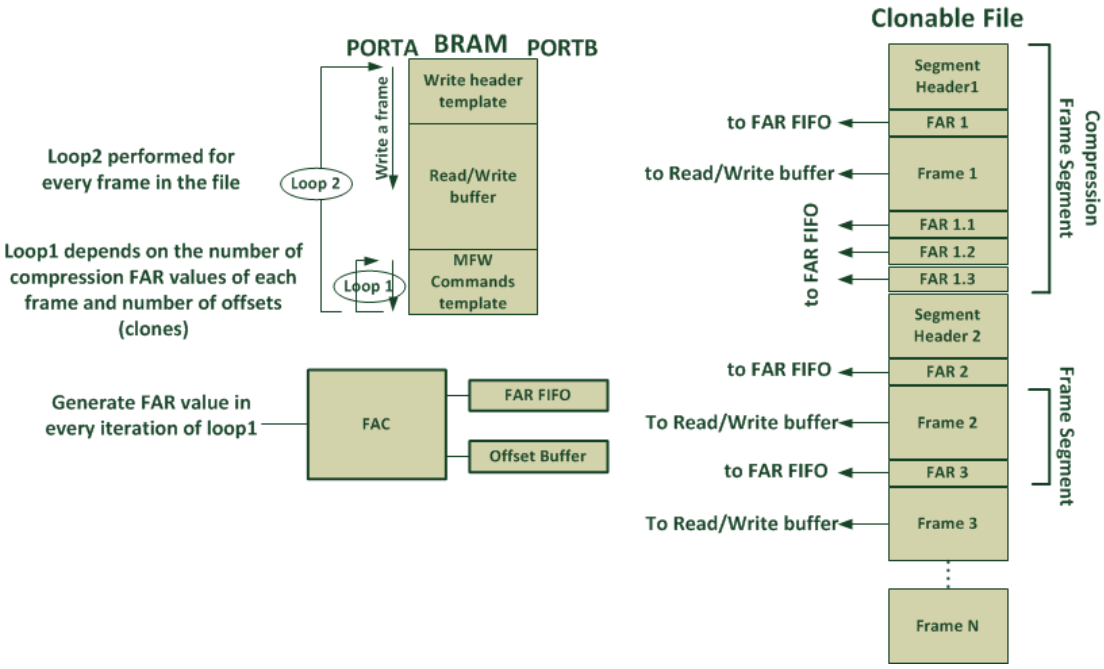



**Figure 4.16 Configuration using the clonable bit file**

## 4.5 Performance and Resource Utilisation Evaluation

This section of the thesis evaluates the resource utilisation of the presented ICM as well as the performance achieved when carrying out the different configuration operations. The presented ICAP controller was implemented in aVirtex-4 FPGA. The system was configured as a standalone ICAP controller where the soft-processor is responsible for initiating different test operations without the need for a main CPU.

### 4.5.1 Resource Utilisation Evaluation

The full ICM was designed to support many operations and to be the core hardware component in an ROS. Not all reconfigurable systems require the capabilities of the presented ICM. The ICM design allows for easily removing any unwanted feature at the RTL design stage in order to reduce the resource utilisation.

Table 4.6 shows the resource utilization of the different components in the full ICM whereas Table 4.7 shows the total resource utilisation for different versions of the ICM with different sets of supported features.

The full ICM utilises 1,117 slices in a Virtex-4 FPGA. This accounts for around 4% of the largest device in the Virtex-4 family (XC4VFX60) and around 20% for the smallest device in the family (XCVFX12). The resource utilisation of the ICM depends on the features required in a particular system. For an ROS system that requires module relocation as well as reading/writing individual frames, trimming the extra features of the ICM results in a resource utilisation of 721 slices. When only basic DPR is required, no soft-processor is required as controlling the configuration becomes a simple process requiring a small FSM utilising only 74 slices and a single BRAM block. It is noted that the resource utilisation figures reported in Table 4.7 are given when an SRAM module is used to store the partial bitstreams. A secondary NPI-MPMC controller was also designed to support configuration from DDR memories [137]. The controller resource utilisation is 107 slices; however, it connects to the Xilinx MPMC IP, which utilizes around 1,250 slices and 2 BRAMs

in a Virtex-4 FPGA. Thus, the NPI-MPMC controller is only suitable for large FPGAs and systems that already use a DDR memory module. It is also noted that the current version of the DDR memory controller does not support the multiple-clone configuration feature. In addition, relocation is only possible when a module is relocated to a location within the same FPGA half as the original location of the module. This means that two bitstreams are required for each relocatable module. The first is generated for the top half of the FPGA whereas the second is generated for the bottom half of the FPGA.

**Table 4.6 ICM's resource utilisation in a Virtex-4 FPGA**

| Component | Slices | BRAM |
|---|---|---|
| ***Pico-blaze*** | 104 | 1 |
| ***ICAP Controller*** | 803 | 1 |
| ***SRAM Controller*** | 146 | 0 |
| ***Glue Logic*** | 64 | 0 |
| ***Total*** | 1117 | 2 |

**Table 4.7 Resource utilisation for different versions of the ICM**

| Feature | | | | | | Resource Utilisation | |
|---|---|---|---|---|---|---|---|
| ***DPR (No Pico-blaze)*** | ***Frame Read/write*** | ***Relocation*** | ***Blanking*** | ***Cloning*** | **Online CRC** | ***Slices*** | ***BRAM*** |
| YES | NO | NO | NO | NO | NO | 75 | 1 |
| YES | YES | NO | NO | NO | NO | 417 | 2 |
| YES | YES | YES | NO | NO | NO | 721 | 2 |
| YES | YES | YES | YES | NO | NO | 796 | 2 |
| YES | YES | YES | YES | YES | NO | 992 | 2 |
| YES | YES | YES | YES | YES | YES | 1117 | 2 |

The fact that the proposed ICM can perform and control all the configuration operations independently means that it is possible to configure the ICM as a standalone configuration system for applications that do not require an embedded host processor. This capability is not possible in systems based on the Xilinx

HWICAP as it depends on a host processor to control its operation and pass the configuration data through the host bus. Such systems can incur additional resource overheads caused by all the required components (e.g. bus controller, DMA). Table 4.8 compares the resource utilisation of the proposed ICM with two different implementations of a HWICAP based system [88].

**Table 4.8 Resource utilisation comparison between proposed ICM and HWICAP based systems in a Virtex-4 FPGA**

| *Configuration System* | *Resource Utilisation* | |
|---|---|---|
| | *Slice* | *BRAM* |
| ***Proposed ICM (SRAM Interface)*** | 1117 | 2 |
| ***HWICAP without DMA [88]*** | 1637 | 0 |
| ***HWICAP with DMA [88]*** | 2138 | 0 |

## 4.5.2 Standard Configuration Operations Performance Evaluation

The standard configuration operations are: configuration frames read/write and basic DPR. Although some authors suggest overclocking the ICAP to achieve higher configuration throughput (see Chapter 3), the overclocking technique is not considered in the performance analysis, despite the fact that the proposed ICM has operated successfully with a frequency of up to 160 MHz (60Mhz higher than the ICAP rated frequency). This is mainly because of the following reasons:

1) Results obtained from overclocking the ICAP may not be applicable to all designs and all devices. Variation in devices and experimental conditions will directly impact the maximum throughput achieved.
2) There might be some reliability issues when overclocking the ICAP. It is difficult to predict how the ICAP will behave in the long run when overclocked. This is especially important as dynamic reconfiguration is often used for enhanced reliability.

3) There is a risk of damaging the internal configuration logic when operating the ICAP with much higher clock frequencies compared to the maximum rated frequency.

Due to the aforementioned reasons, the performance analysis is based on how close the achieved configuration throughput is to the maximum theoretical throughput. Table 4.9 shows the time measurements for the basic read/write configuration operations. To test the configuration speed of the proposed ICM, three benchmark RMs that span different areas on the FPGA are considered (see Table 4.10). Two partial bitstreams are generated for each RM; the first is generated without compression; resulting in the maximum file size whereas the second is for a black-box of the same area as the RM. The black-box is generated with compression enabled, resulting in the smallest file size.

**Table 4.9 Frame read/write time overhead**

| Operation | ICAP Freq. (MHz) | Operation Time (us) |
|---|---|---|
| ***Read Frame*** | 100 | 2.28 |
| ***Write Frame*** | 100 | 1.95 |

**Table 4.10 Benchmark RMs**

| RM ID | RM Area (Columns) | | | File Size (KB) | |
|---|---|---|---|---|---|
| | CLB | BRAM | DSP | Normal | Black-Box (Compressed) |
| ***RM1*** | 2 | 0 | 0 | 8 | 3 |
| ***RM2*** | 4 | 1 | 0 | 29 | 7 |
| ***RM3*** | 8 | 1 | 1 | 47 | 10 |

Table 4.11 shows the configuration times and the relocation times for the benchmark RMs using the default SRAM controller and the secondary NPI-MPMC controller when operating at 100MHz and using the 32-bit configuration of the ICAP.

**Table 4.11 Configuration and relocation times of the ICM**

| RM ID | SRAM Controller | | | | NPI-MPMC Controller (DDR2) | | | |
|---|---|---|---|---|---|---|---|---|
| | Configuration Time (us) | | Relocation Time (us) | | Configuration Time (us) | | Relocation Time (us) | |
| | Normal | Black-Box | Normal | Black-Box | Normal | Black-Box | Normal | Black-Box |
| ***RM1*** | 21.08 | 8.28 | 21.17 | 8.73 | 30.32 | 11.42 | 30.41 | 11.83 |
| ***RM2*** | 74.84 | 18.52 | 74.95 | 20.27 | 109.28 | 26.56 | 109.39 | 28.31 |
| ***RM3*** | 120.92 | 26.20 | 121.04 | 29.05 | 176.96 | 37.84 | 177.08 | 40.69 |
| ***Avg. Throughput*** | 376.2 (MB/s) | 365.2 (MB/s) | 375.4 (MB/s) | 336.3 (MB/s) | 258.8 (MB/s) | 257.3 (MB/s) | 258.3 (MB/s) | 243.4 (MB/s) |

From Table 4.11, we can see that the maximum configuration throughput using the SRAM controller is very close to the maximum ICAP throughput, which is 400 MB/s. Furthermore, the configuration throughput increases as the size of the partial bitstream increases. When operating the ICM with an NPI-MPMC controller, the configuration throughput degrades due to the latency of each burst transfer from the DDR memory. RM relocation slightly reduces the configuration throughput. In the case of the SRAM controller, the relocation throughput of non-compressed bitstreams is 0.8 MB/s slower than configuration without relocation. The relocation overhead depends on the number of FAR values present in the partial bitstream as substituting a new FAR value requires a single clock cycle in the proposed ICM. For compressed partial bitstreams, the number of FAR values depends on the number of identical frames in the partial bitstream. A black-box will result in the maximum number of FAR values in the partial bitstream of any RM. For the benchmark RMs, the average relocation throughput of the black-box bitstreams was 336.3 MB/s. This is 39.1 MB/s less than the average relocation throughput of the non-compressed bitstreams. We can estimate the relocation throughput of proposed ICM by: (average relocation of non-compressed bitstreams + average relocation throughput of compressed black-box bitstreams)/2. This results in an average relocation throughput of 355.9 MB/s in the case of the SRAM controller and 250.9 MB/s in the case of the

NPI-MPMC controller. Table 4.12 compares the relocation throughput of the proposed ICM with the throughput of different relocation systems reported in the literature.

**Table 4.12 Throughput comparison between the proposed ICM and other relocation systems**

| System | Device | ICAP Freq. (MHz) | Throughput (MB/s) | Proposed ICM Speedup | |
|---|---|---|---|---|---|
| | | | | SRAM | NPI-MPMC |
| ***REPLICA2Pro [49]*** | Virtex-II/Pro | 35 | 35 | x10.2 | x7.7 |
| ***BiRF [50]*** | Virtex-4 | 100 | 7.3 | x48.9 | x34.4 |
| ***[47]*** | Virtex-4 | 100 | 3.5-8.9 | x101.7-x40.0 | x71.7-x28.2 |
| ***ARC [138]*** | Virtex-4 | 100 | 61.9 | x5.7 | x4.1 |
| ***OORBIT [139]*** | Virtex-4 | 100 | 100 | x3.6 | x2.5 |

Table 4.12 shows that the proposed ICM outperforms the relocation systems reported in the literature. The relocation system providing the closest throughput to the proposed ICM is the OORBIT, which was implemented in a Virtex-4 FPGA. The proposed ICM was 3.6 times faster with the SRAM controller and 2.5 times faster with the NPI-MPMC controller. In fact, the OORBIT is not a pure online bitstream relocation system as it uses pre-computed offline FAR and CRC values generated for all the possible locations of the RMs to accelerate the relocation process.

The REPLICA2Pro and BiRF are early relocation filters, which perform the FAR modifications online using dedicated hardware. The reported throughput of REPLICA2Pro is just an estimation provided by the authors and is not based on an actual implementation. The BiRF is also a hardware relocation filter with an average throughput of 7.2 MB/s when operating at 100 MHz. The BiRF poor throughput is mainly because the partial bitstreams were fetched from the DDR memory using a processor bus rather than a DMA engine. The system in [47] uses software running on a soft-processor to perform the partial bitstream modification required for relocation. Using software to modify the partial bitstream before configuration degrades the relocation throughput especially when bitstream reversal is required

(i.e. moving a bitstream from top half of the FPGA to the bottom half or vice versa). ARC also uses software to perform some of the bitstream modifications. However, bitstream reversal is performed in hardware to avoid the delay caused by rearranging the configuration frames in software prior to configuration.

### 4.5.3 Online Black-Box Bitstream Generation

The ICM can generate a black-box bitstream online for a relocatable RM with any size. The main aim of this feature is to circumvent the need for storing extra black-box partial bitstreams, which are required for the removal of already configured RMs in the system. The RM removal process is based on configuring compressed empty columns covering the RM area. Table 4.13 shows the removal time of the benchmark RMs shown in Table 4.10 using the proposed removal method, non-compressed black-boxes and compressed black-boxes.

It can be seen from Table 4.13 that the RM removal time of the proposed online method falls between the removal time of the non-compressed black-box file configuration and the compressed black-box file configuration. The proposed method does not require the storing of any files externally opposite to the other two methods. For very small RMs the proposed method might be faster than the other two methods as a black-box file generated using the BitGen tool contains a fixed header and a trail sequence of NOPs, adding extra configuration overhead.

**Table 4.13 RM removal time using (SRAM controller)**

| RM ID | Removal Time (us) | | |
|---|---|---|---|
| | Non-Compressed Black-Box | Compressed Black-Box | Proposed Method |
| ***RM1*** | 21.17 | 8.73 | 7.46 |
| ***RM2*** | 74.95 | 20.27 | 26.38 |
| ***RM3*** | 121.04 | 29.05 | 44.93 |

### 4.5.4 The Multiple-Clone Configuration Technique

This feature allows for copying the same relocatable module in different locations on the FPGA in a single operation and using a single clonable bitstream file. To see how the proposed multiple-clone configuration method would scale compared to the normal configuration method, two relocatable modules are used for testing. The first module is a small Pulse Width Modulation (PWM) module and the second module is a large K-means clustering core with fixed 8 clusters and a dataset of 2,905 points. The K-means clustering core is based on the implementation discussed in [140]. Table 4.14 shows the resource utilisation and area occupation of the two modules.

**Table 4.14 Test relocatable cores**

| Test Core | Resource Utilisation | | Area (Columns) | | Bitstream Size (KB) | Clonable File Size (KB) |
|---|---|---|---|---|---|---|
| | *Slices* | *BRAM* | *CLB* | *BRAM* | | |
| ***PWM*** | 46 | 0 | 2 | 0 | 8 | 7.4 |
| ***K-Means*** | 1107 | 5 | 16 | 2 | 85 | 87.4 |

Table 4.14 shows that the size of the clonable file is comparable to the size of the original non-compressed bitstream generated by the BitGen tool. It is slightly smaller than the bitstream for the small PWM module and slightly larger than the bitstream for the K-means core. Table 4.15 shows the configuration times for configuring multiple instances of the test cores within different locations in the same half of a Virtex-4 FPGA (XC4VFX60). The configuration locations are limited to the same half of the FPGA because bitstream reversal in not possible with the multiple-clone configuration as each configuration frame is written to the FDRI register before writing the MFW commands. If locations spanning both halves of the FPGA are required, two multiple-clone operations must be performed: one for configuring the relocatable module on the top half locations and the other for the locations in bottom half of the FPGA.

**Table 4.15 Configuration times of the test cores**

| Configuration Method | Configuration Times for Different Number of Instances (ms) | | | | | |
|---|---|---|---|---|---|---|
| | K-means | | | PWM | | |
| | 4 | 5 | 6 | 10 | 15 | 20 |
| ***Normal*** | 0.87 | 1.1 | 1.3 | 0.21 | 0.31 | 0.42 |
| ***Multiple-Clone*** | 0.40 | 0.44 | 0.49 | 0.057 | 0.075 | 0.092 |
| ***Speedup*** | x2.2 | x2.5 | x2.7 | x3.7 | x4.1 | x4.6 |

It can be seen from Table 4.15 that the configuration speedup using the multiple-clone configuration technique scales with the number of instances configured. For the K-means core, six instances are possible within one half of the Virtex-4 FX60. The configuration time for configuring the K-means core can be 2.5 times smaller with the multiple-clone configuration technique compared to the normal configuration method. More than 60 instances for the smaller PWM module are possible within one half of the FPGA. When configuring 20 instances of the PWM module, the configuration time of the multiple-clone configuration is 3.8 times smaller than the normal configuration method.

It is important to mention that the multiple-clone configuration is not just limited to the Virtex-4 FPGA family. In fact, the possible gain with this technique is higher for newer devices such as the Virtex-6 and the 7-series families as the size of a configuration frame in these families is larger compared to the Virtex-4. For example, the number of words in a Virtex-6 FPGA is 81 compared to the 41 words in a Virtex-4 FPGA and still can be cloned in the seven clock cycles needed for the MFW command sequence. This means that a reduction in configuration time is around 87% for a cloned frame in a Virtex-6 FPGA compared to 76% in a Virtex-4 FPGA.

## 4.6 Chapter Conclusion

In modern FPGAs, access to the configuration memory from within the FPGA allows for the implementation of interesting self-reconfiguring systems. The limited configuration throughput of FPGAs can be a performance bottleneck, especially for

systems that extensively use the configuration port for complex operations such as online bitstream relocation. The high configuration throughput required for such systems drives the need for an efficient configuration management system that independently handles these operations without degrading the configuration throughout.

This chapter presented the design and architecture of an ICM that supports all the basic configuration operations with minimal overhead. In addition, the ICM supports a set of specialised features such as bitstream relocation and the multiple-clone configuration technique. The relocation throughput of the presented ICM is superior to the pre-existing relocation systems allowing for throughputs very close to the maximum limit of the configuration port. Moreover, the proposed multiple-clone configuration technique can achieve configuration throughputs multiple times higher than the theoretical limit of the ICAP without overclocking the configuration port when multiple instances of the same relocatable model are configured. The comprehensive feature support and the high performance of the presented ICM make it especially suitable for implementing complex reconfigurable applications such as an ROS, which is extensively affected by the configuration performance.

# Chapter 5 : Reliability-Centric Internal Configuration Management

The internal dynamic reconfiguration feature in modern FPGAs has opened the door for new opportunities to implement low-cost self-healing systems. Many techniques for online fault detection and repair in FPGAs have been proposed in the literature (see Chapter 4). In order to realise a fully autonomous single-chip solution, the fault detection and repair should be carried out from within the FPGA using the internal configuration port. Self-healing systems require the ICM to perform different fault detection and repair operations on-the-fly without disturbing the implemented system. These operations should be performed at the highest speed possible, especially when the internal configuration port is extensively used for operations other than fault detection and repair. The performance of the system could be degraded when access to the configuration port is dominated by fault detection and repair. This brings the need for an efficient management of all operations requiring access to the configuration port. In addition, the ICM should be designed to withstand faults in its logic as configuration errors could lead to system failure. The design of the ICM should be compact as the resource overhead of the conventional fault-mitigation design techniques could impact the overall efficiency of the system.

The work presented in this chapter presents different fault detection and repair schemes that can be performed by the ICM to mitigate soft faults such as internal readback scrubbing, external scrubbing and CRC-based configuration verification. This chapter also discusses different design strategies to ensure reliable operation of the ICM. The R3TOS is also presented in this chapter as a solution to handle permanent faults in the FPGA by means of bitstream relocation. The R3TOS aims to integrate all the fault detection and repair capabilities of the ICM in an ROS that provides a generic platform for FT applications.

# 5.1 The Design of a Fault-Tolerant ICM

When used for FT applications, the ICM should be able to perform different fault detection and repair operations. The design of the ICM should be able to tolerate faults in its logic in order to perform these operations correctly and prevent system failure. The high performance ICM discussed in Chapter 4 was redesigned with different DPR-based self-recovery techniques to come up with the optimal design in terms of performance and area overhead.

## 5.1.1 Triple Modular Redundancy (TMR)

TMR provides great fault detection capabilities. Any fault that affects the output of the ICM is detected by output comparators. In addition, the operation of the ICM is not affected if faults do not occur in more than one redundant module. The main drawback of TMR is the large resource overhead due to triplicating the logic. There are two main criteria in designing the TMR system for the ICM. The first is to reduce the possibility of single faults affecting more than one redundant module in the system. The second is the ability to recover from faults affecting any redundant module.

Faults that affect more than one module are usually caused by a change in the routing between two modules. This problem is apparent in TMR systems with the logic of the three redundant modules placed in the same area. One solution to this is to place the logic of each redundant module in a distinct area with all local routes constrained to this area. These regions could be reconfigurable regions with partial bitstreams generated for recovery by partial reconfiguration. Recovery by partial reconfiguration covers all types of soft error in the configuration memory. In order to reduce the overall area occupation of the TMR system, the minimal number of reconfigurable regions should be used. A large grain TMR scheme uses three reconfigurable regions, one for each instance of the ICM (see Figure 5.1). The three instances should be synchronised at all times and should be able to automatically recover from any faults in one of the three instances. When one of the tree instances

fails, the remaining two instances enter a recovery state after finishing the current configuration operation. In the recovery state the faulty module is reconfigured. Dedicated reset logic resets the three instances to the initial state after recovery and system operation is resumed normally.

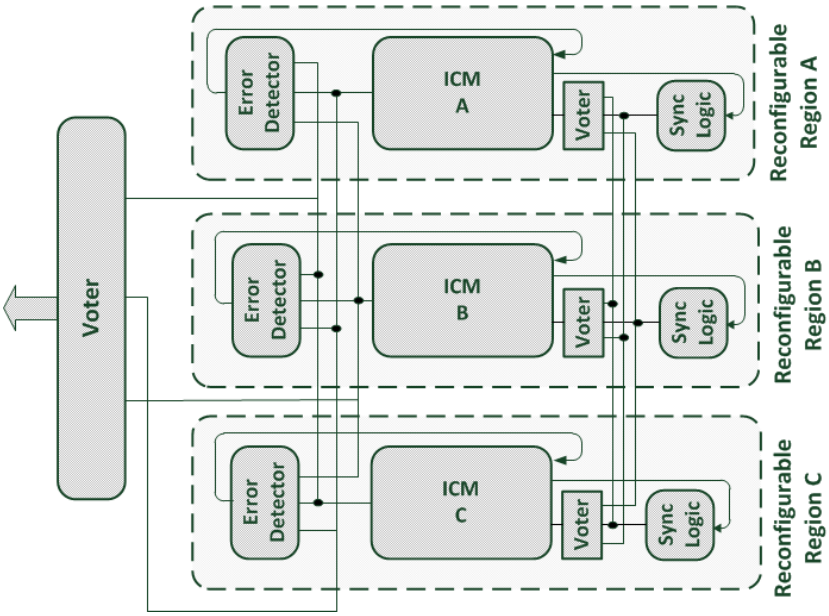



**Figure 5.1 TMR design for the ICM**

## 5.1.2 Dual Modular Redundancy (DMR)

In smaller FPGAs, triplicating the ICM might cause intolerable resource utilisation. DMR requires two redundant modules. One is the default module and the other is used for error detection. When a mismatch between the two modules is detected, the operation is aborted, and the two modules are reconfigured and reset to the initial state.

While DMR reduces the resource utilisation of TMR by 1/3, it is not applicable to the ICM in its basic form. When one of the two instances fails, there is no mechanism to determine which instance is faulty and which instance should gain access to the ICAP to carry out the recovery process. DMR can still be applied to the ICM by implementing a third basic recovery controller, which is only used to

reconfigure the two redundant modules when a mismatch is detected (see Figure 5.2). This is a very basic DPR operation performed by moving the partial bitstream from external memory to the ICAP. The basic recovery controller can be more than 90% smaller than the full ICM. Given the small footprint of the recovery controller, TMR is applied to the recovery controller, which is placed in a separate reconfigurable region. In case of a fault in the recovery controller during the recovery of the ICM, TMR will filter out any error. When the ICM is fully reconfigured, access to the ICAP is switched to the ICM, which immediately reconfigures the region containing the recovery controller to prevent accumulation of faults.

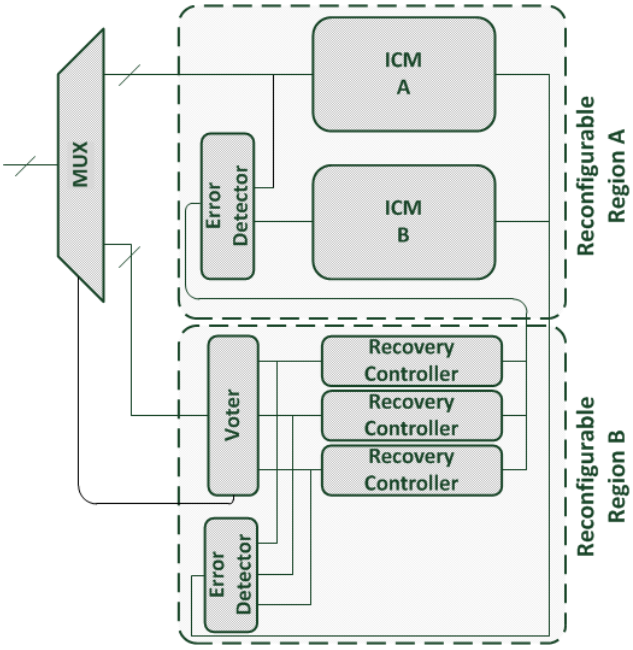



**Figure 5.2 DMR design for the ICM**

## 5.1.3 Operation Monitor

The TMR and DMR implementations of the ICM prevent the passing of faulty data to the configuration memory at the cost of tripling or doubling the resource utilisation of the ICM. The ICM itself is equipped with CRC-32 detection capabilities, which detect faults after performing any configuration write operation

by comparing a checksum generated by the CRC-32 generator with the checksum generated by the FPGA's internal configuration circuitry (see Chapter 4). There are two failure scenarios in ICM when performing any configuration operation: the first is passing faulty data to the configuration memory and the second is failing to complete the configuration operation. In the first scenario, the fault can be detected by CRC checksums, whereas in the second scenario the ICM will stall and not reach the stage of the CRC check. As each configuration operation has a deterministic time, an operation monitor can be implemented to check if the ICM stalls at any operation or a CRC error is detected. The operation monitor can trigger the recovery controller to reconfigure the ICM in any of the two failure scenarios (see Figure 5.3).

This thesis proposes using the operation monitor to circumvent the need for replicating the ICM to reduce the overall resource utilisation. However, errors in configuration are detected after they take effect in configuration memory, which is not the case for TMR and DMR.

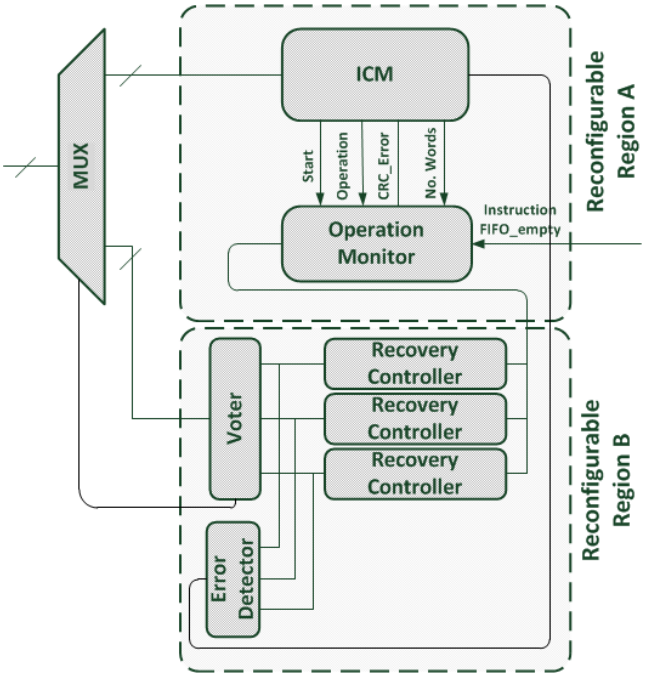



**Figure 5.3 CRC error detection in the ICM**

### 5.1.4 Resource Utilisation vs. Performance

The resource utilisation for the different FT versions of ICM in a Virtex-4 FX60 FPGA are shown in Table 5.1. Because the designs are based on DPR, the area of occupation is also considered. Table 5.2 shows the area occupation and the maximum recovery time of each version of the FT ICM. The maximum recovery time is defined as the reconfiguration time of the region containing the full ICM. The TMR design was floor-planned so that each reconfigurable region is placed in a single clock region of the Virtex-4 FX60 FPGA (total clock regions =16). In the DMR design, the ICM's region occupies two clock regions, whereas the TMR-ed recovery controller occupies half the clock region. In the proposed small design, the ICM's region is placed in a clock region and the TMR-ed recovery controller is placed in a half clock region.

**Table 5.1 Resource utilisation of different versions of the FT-ICM in a Virtex-4 FPGA**

| Resource Type | TMR | DMR | Proposed (Operation Monitor) |
|---|---|---|---|
| ***Slice*** | 3547 | 2626 | 1513 |
| ***BRAM*** | 6 | 4 | 2 |

**Table 5.2 Area occupation and recovery time in a Virtex-4 FX60 FPGA**

| *Design Strategy* | *No. of RPs* | *No. of Occupied Columns* | *Max. Recovery Time (100MHz)* |
|---|---|---|---|
| ***TMR*** | 3 | 18.8% | 0.40 ms |
| ***DMR*** | 2 | 13.0% | 0.77 ms |
| ***CRC Detection*** | 2 | 6.8% | 0.40 ms |

A fault injection experiment was carried out to estimate the reliability of different versions of the FT-ICM. The fault injection was performed using an online fault injector (a dedicated ICAP controller). Hard-macros for the critical components used in the designs were created. Single faults were injected in the area covered by these components. After each fault is injected, the outputs of the tested component are compared to a reference component before reconfiguration (Table 5.3). All errors in the ICM were detected by TMR, DMR and CRC detection schemes. However,

single points of failure are increased in DMR and CRC schemes as they contain more static components (ICAP multiplexer, error detector).

**Table 5.3 Errors in critical design components**

| Component | No. of Faults Injected | No. of Errors | Error Type |
|---|---|---|---|
| ***ICM*** | 635008 | 35450(6%) | Correctable |
| ***ICAP Voter*** | 57728 | 591(1%) | Uncorrectable |
| ***ICAP Mux*** | 28864 | 316(1%) | Uncorrectable |

## 5.2 Soft-Error Handling Strategies

In the context of this thesis, soft errors are defined as correctable errors that appear as single bit-flips or multiple bit-flips in the FPGA's configuration memory. Configuration memory scrubbing is the process of correcting bit-flips in memory caused mainly by radiation effects. In SRAM-FPGAs, configuration memory scrubbing can be classified into two types: readback scrubbing and external scrubbing (see Chapter 3). Both types can be managed internally using an ICM implemented in the FPGA logic. Readback scrubbing is based on reading the configuration frames using the internal configuration port and performing error checks based on the parity bits embedded in the device's configuration memory. External scrubbing, on the other hand, uses a reference bitstream stored in an external memory to correct any emerging faults in the configuration of the FPGA.

### 5.2.1 Internal Readback Scrubbing

In Xilinx Virtex FPGAs, ECC bits are embedded in the configuration memory. These bits are calculated when a bitstream is generated for a particular design implementation to enable detection when a bit-flip occurs after the device is configured. In the Virtex-4 family, there are 12 Hamming parity bits located in the 21$^{st}$ word of each frame. These bits can be used to detect and correct single-bit errors within a configuration frame. Correction of any corrupted configuration frame is not possible using the ECC when more than one bit-flip occurs in the frame. Error

detection is also limited to two bit-flips within a frame. When more than two bit-flips occur in a frame, there is no guarantee that these errors will be detected [25].

Xilinx has also included a dedicated hard-wired ECC logic block in the Virtex-4 FPGA. This ECC block is connected internally to the ICAP and activated during configuration memory readback (see Figure 5.4). During readback, the ECC block decodes the configuration frames to determine if a bit-flip has occurred. If a faulty frame is encountered during a readback operation, an error signal is raised and a 12-bit syndrome value denoted by S [11:0] is generated by the ECC block. The syndrome bits can be used to determine the type and location of errors within a faulty frame. Table 5.4 shows how errors are classified using the syndrome.

**Table 5.4 ECC syndrome decoding**

| Syndrome | | Error Type |
|---|---|---|
| S [11] | S [10:0] | |
| 0 | = 0 | No error |
| 1 | ≠ 0 | Single-bit error |
| 1 | =0 or $2^n$ | Single error in the parity bits |
| 0 | ≠ 0 | Double-bit error |

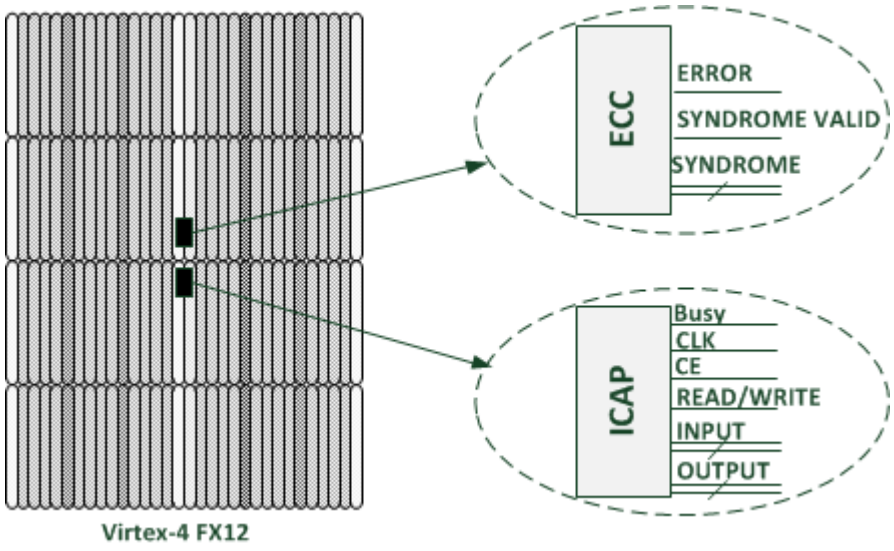



**Figure 5.4 ECC logic block in a Virtex-4 FX12 FPGA**

A Virtex-4 configuration frame contains 41 words, each of 32-bit, giving a total of 1,312 bits indexed in the range of [0:1311]. When a single bit-flip is detected, the syndrome points to the flipped bit using an address space from 704 to 2,047. Figure 5.5 shows the bit indexing and the syndrome indexing for a configuration frame. When the bit-flip is not in the parity bits of the frame, the bit index of the error can be calculated from the value of S [10:0] according to Algorithm 5.1.

***IF*** *S [10:0] < 1024*
    **Error Index** = *S [10:0] – 704*
***ELSE***
    **Error Index** = *S [10:0] – 736*
***END IF***

**Algorithm 5.1 Error index calculation when S [11] = 1 and S [10:0] ≠ 0**

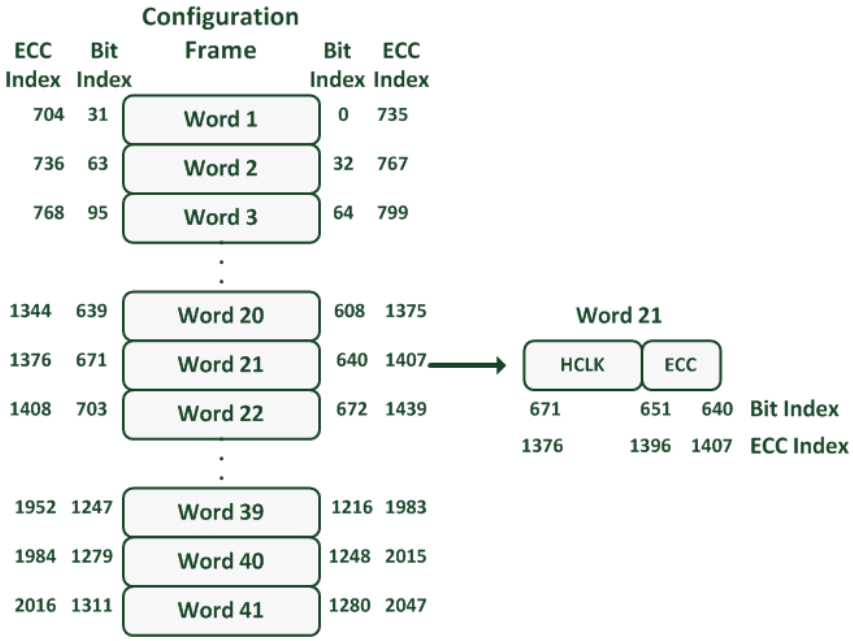



**Figure 5.5 Bit indexing in a configuration frame**

If the detected error is in the parity bits, then the error's bit index will be in the range of 640-651, which is the location of the parity bits within a frame. To locate a faulty parity bit, Algorithm 5.2 can be used.

```
IF S [10:0] = 0
        Error Index = 651
ELSE
        Error Index = 640 + log2 S [10:0]
END IF
```

**Algorithm 5.2 Error index calculation when S [11] = 1 and S [10:0] = (0 or $2^n$)**

To enable fast detection when a large amount of frames are read in a single readback operation, the ECC logic contains an 'error' signal and a 'syndrome valid' signal. The error signal is high when any type of error is detected and the syndrome valid signal stays high for one clock cycle at the end of each frame. The ECC readback scrubbing scheme cannot be used to protect the BRAM resources in the device. In addition, LUTs configured as SRL16 or distributed RAM must be masked during the scrubbing process; otherwise the syndrome would be corrupted by the dynamic values in the LUTs. The readback options allow for choosing whether to include the LUTs contents in the readback data or the initial values used for the parity bits calculation. This can be accomplished by setting the BLUTMASK_B to '0' in the control register [25].

To scrub a configuration frame, the ICM operation can be divided into the following steps: first the ICM initiates a 'read' operation for the frame to be checked. This frame is stored in the ICM's read/write buffer. The error signal is checked at the end of the read operation. If an error is detected, the syndrome is registered and the location of the error is extracted from the syndrome by the ICM's soft processor. The final stage is to flip the faulty bit in the read/write buffer before writing the frame back to the same address.

During readback, the data passed to the ECC logic is one clock cycle ahead of the data available on the output port of the ICAP (see Figure 5.6). This data misalignment means that when the readback process is aborted at a particular frame address, the ECC logic will contain the first word of the next frame address. In many situations, not all of the area in the FPGA is required to be scrubbed. In this case, the readback process is divided into different operations covering different areas of the

configuration memory. As there is no reset function for the ECC logic, when readback is resumed at a different frame address, the ECC logic will resume the calculation of the syndrome from the second word of the frame, making wrong syndrome calculations for all the consecutive frames.

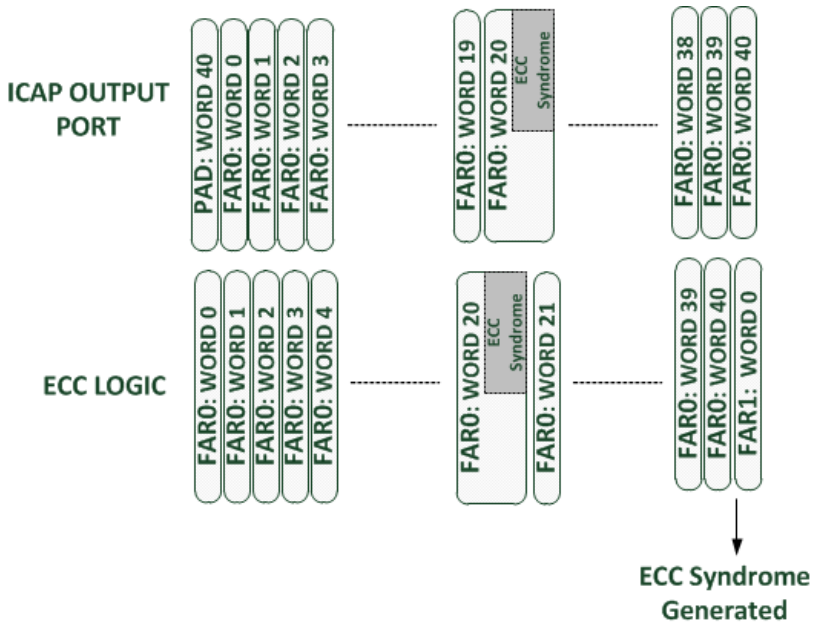



**Figure 5.6 Configuration data mismatch between ICAP and ECC logic**

There are two solutions proposed for this problem. The first is to read a dummy 40 words from the configuration memory and discard the last syndrome at the end of each scrubbing readback operation (see Figure 5.7). This is suitable when the area covered by a single readback operation is large; however, this will significantly increase the time required to finish scrubbing when frames are read individually. In addition, not all the read operations performed by the ICM are intended for the purpose of scrubbing; these read operations will still cause the activation of the ECC block, so the first solution is not time efficient when the system demands a large number of non-scrubbing read operations. The second solution is to ignore the synchronisation between the ICAP and the ECC logic when performing non-scrubbing read operations; however, a count of the number of operations is kept in a

register. When the number of read operations reaches 40, the register is reset and the process is repeated. Before the ICM switches to a scrubbing read operation, a dummy read operation of (40-count) is performed to synchronise the ICAP and the ECC logic.

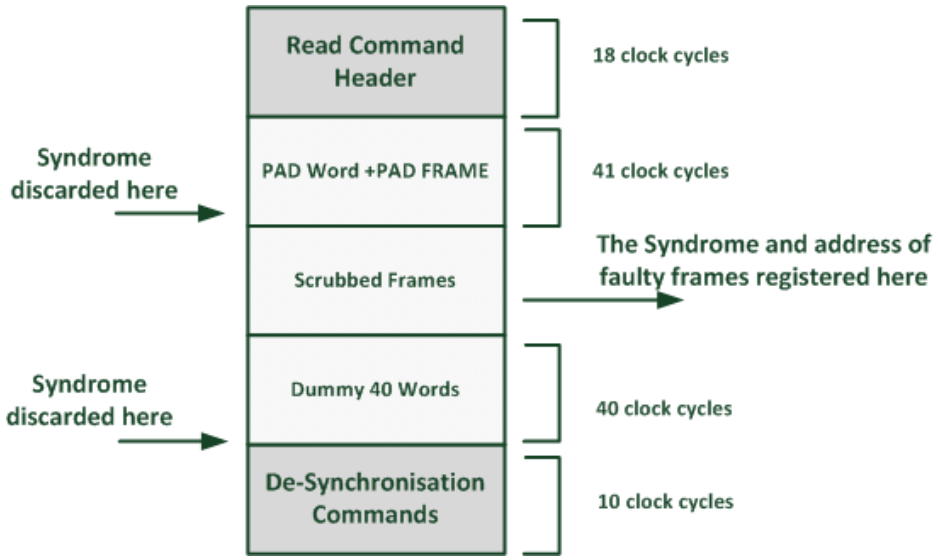



**Figure 5.7 The ICM scrubbing read operation**

## 5.2.2 External Configuration Memory Scrubbing

External scrubbing does not use the embedded ECC parity bits; instead it uses a golden bitstream stored in the external memory. External scrubbing can correct any number of bits within a frame, provided that these bits are not the configuration bits of BRAM resources and LUTs configured as SRL16 and distributed RAM. The golden bitstream is a modified version of the original bitstream with all BRAM configuration frames removed. Similar to readback scrubbing, writing to LUTs configured as SRL16 and distributed RAM should be disabled by setting the GLUTMASK_B bit to '0' in the control register before each write operation.

There are two strategies for external scrubbing. The first is the 'event-triggered' scrubbing, whereby the scrubbing operation is only triggered when a fault is detected. For example, if a redundancy system detects a fault in a redundant module,

external scrubbing is triggered to correct the fault. Opposite to fault correction by DPR, external scrubbing does not reset the internal registers of the faulty modules. The second strategy for deploying external scrubbing is referred to as 'blind' scrubbing. In blind scrubbing the configuration of the FPGA is periodically refreshed, even if no faults have affected the system.

Usually, an external scrubbing strategy will use one of the external configuration ports of the FPGA and some external circuitry to control the scrubbing process [114]. In some systems the external golden bitstream is loaded to the configuration memory using the internal configuration port [117]. To protect the golden bitstream from radiation induced faults, the golden bitstream is usually stored in a radiation hardened non-volatile memory which is especially fabricated to tolerate high levels of radiation. In many cases, the control circuitry of the external memory module is implemented on the FPGA logic. This control logic interfaces to the external memory module using the IOs of the FPGA. The speed of external scrubbing can be affected by the latency and maximum throughput of the memory module used to store the golden bitstream. To achieve maximum scrubbing speed, the ICAP should operate at 100MHz; this could be challenging for non-volatile memory modules. It is a common practice to move the bitstream from a non-volatile memory to faster SRAM or DRAM external memory modules after power-up of the device to enable fast reconfiguration. A golden bitstream is assumed to be fault-free, when the bitstream is stored in an SRAM or DRAM memory module; the assumption of a fault-free bitstream does not hold as these memories are susceptible to soft errors. Parity bits must be used to detect errors in the memory. It is not necessary for the parity bits to provide correction capabilities as the original bitstream will be stored in a non-volatile memory and will always be available for correction.

In Xilinx Virtex FPGAs, there is an internal CRC generator used for configuration verification (see Chapter 3). The embedded CRC can be used to detect faults in the reference bitstream when performing external scrubbing. However, faults will only be detected after loading the bitstream into the FPGA's configuration memory. This is critical especially when burst errors occur as a result of faults in the memory interface. To reduce the possibility of burst errors corrupting a large number of

configuration frames, the scrubbing operation can be performed in a frame-based scheme, whereby frames are configured individually using separate write operations [112].

When writing frames individually to the configuration memory, an extra PAD frame is written in each operation. This will significantly increase the total time required to complete a scrubbing cycle.

This thesis proposes using the CRC-32 generator of the ICM presented in Chapter 4 to perform the CRC checks during the scrubbing process so that the scrubbing process can be aborted when a fault is detected. This capability requires modifying the ICM to include some external scrubbing control logic that compares CRC values generated by a CRC generator with pre-computed CRC values at fixed intervals during the scrubbing process. The pre-computed CRC values can be stored in an on-chip memory block to accelerate the CRC comparison process (see Figure 5.8). Using this proposed method, scrubbing can be stopped once a fault is detected without increasing the total scrubbing time.

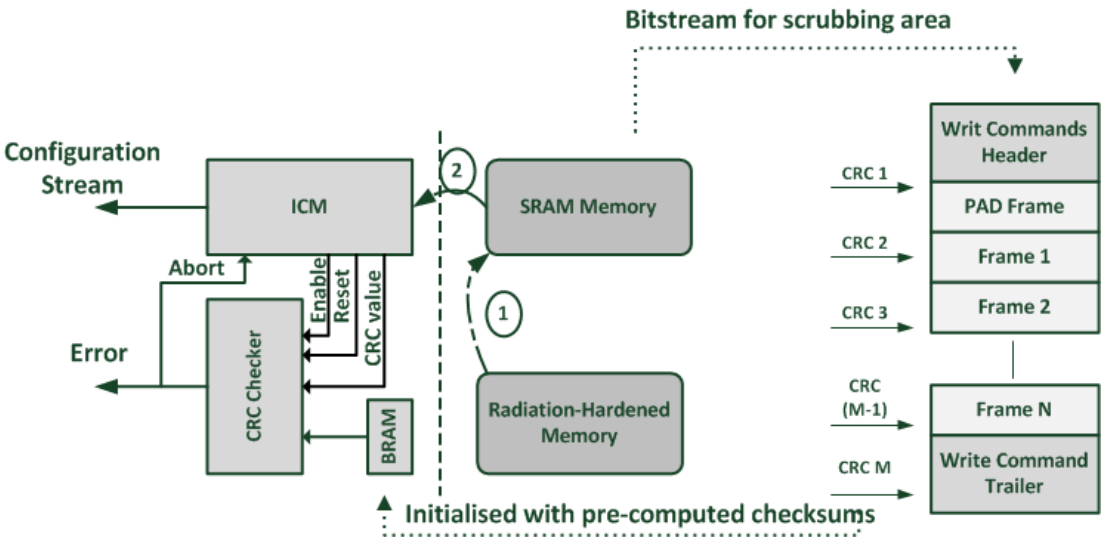



**Figure 5.8 Online CRC for external scrubbing**

The major disadvantage of this method is the memory required for storing the pre-computed CRC values. Considering that not all of the FPGA resources are required

to be covered by external scrubbing, one or only a few BRAMs might be enough to perform scrubbing on the critical areas in the FPGA. A single Virtex-4 BRAM can store 2 KB of data. If a CRC check is performed for each frame, a total of 23 CLB columns can be covered using a single BRAM.

### 5.2.3 Configuration Memory Scrubbing Evaluation

The scrubbing time overhead will depend on the scrubbing area and the ICAP operating frequency. Figure 5.9 shows the scrubbing time for three scrubbing schemes: the first is readback scrubbing, whereby the entire scrubbing area is covered by a single read operation [113]. The second scrubbing scheme is the frame-based scrubbing scheme, whereby individual frames are written separately to the configuration memory [112]. The third scrubbing scheme is the proposed external scrubbing scheme, whereby the scrubbing area is covered by a single write operation.

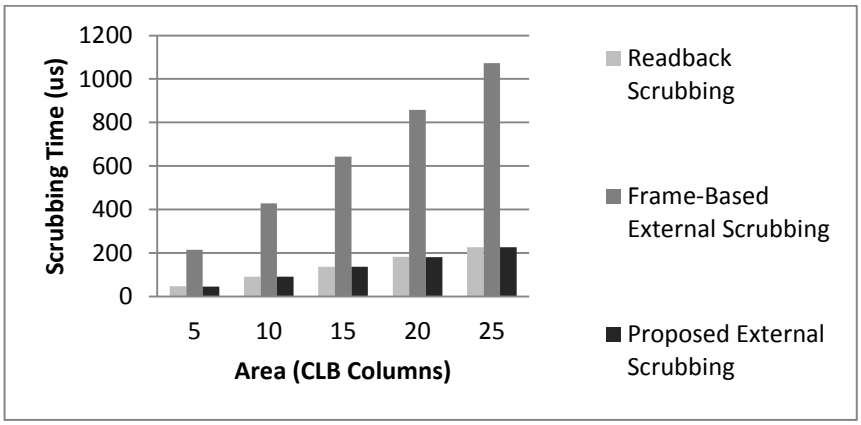



**Figure 5.9 Scrubbing time overhead**

The choice of whether to use readback or external scrubbing in FPGAs will depend on the application used. Internal readback scrubbing does not need any major external circuitry and can be performed using just the embedded ECC block in the FPGA. However, the number of correctable faults within a single frame is limited. This is not the case for external scrubbing which requires an interface to an external

memory module that stores the golden bitstream. There are a number of single points of failure depending on the scrubbing scheme used. In readback scrubbing, faults in the interconnections between the scrubber logic and the ECC logic block can cause the scrubbing system to fail. Similarly, external scrubbing has single points of failure in the interconnections to the IOs connecting the external memory module to the FPGA. A combination of both schemes can reduce the overall number of single points of failure. Supposing that external scrubbing is the primary scrubbing scheme, when a fault is detected by a CRC mismatch between the pre-computed values and the value generated online, a readback scrubbing cycle can be performed to recover any possible faults in the external memory interconnections. In the case of a Virtex-4 FPGA, all single-bit faults can be eliminated (see Figure 5.10).

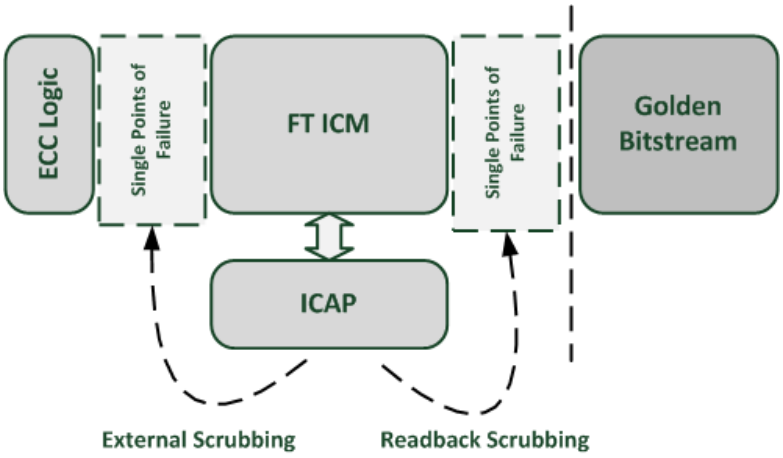



**Figure 5.10 Combined external and readback scrubbing schemes**

When using the ICAP to perform any type of configuration memory scrubbing, the interconnections between the scrubber logic and the ICAP act as single points of failure. Faults in these interconnections can cause system failure because the ICAP will not be accessible for any kind of recovery process. Recovery in the case of faults in the ICAP interface will require reconfiguration using the external configuration port. Recovery can be accomplished by either a full reconfiguration of the whole bitstream or by partial reconfiguration of the region affected. When performing a full reconfiguration, the whole system will be reconfigured and reset to the initial state.

Partial reconfiguration can be used to recover the ICAP without interrupting the operation of the unaffected components. This, however, will require specific placement to ensure that the routing to the ICAP interface is confined within a reconfigured region. One way to do this is to place the ICM in a reconfigurable region that covers the CLB columns adjacent to the ICAP primitive so that all the ICAP routes are inside the reconfigurable region.

## 5.3 Permanent-Fault Handling Strategies

Permanent faults, or hard faults, manifest themselves as irreversible physical defects in the device. Dealing with emerging permanent damage in the resources of the FPGA is more complex than dealing with soft errors as conventional configuration memory scrubbing will not have an effect on the damaged resources. In addition, only stuck-at faults can be detected with readback scrubbing.

In general, permanent faults can be detected by loading specialised Built-In Self-Test (BIST) circuits that perform online testing to identify a faulty resource in a particular area of the FPGA. Permanent faults can be mitigated by rearranging the functional modules in the FPGA so that faulty resources are circumvented and discarded (bitstream relocation).

### 5.3.1 General Fault Mitigation Scheme

One of the major drawbacks of online BIST diagnosis is the time overhead for loading the different BIST circuits. Similar to external scrubbing, BIST diagnosis can be either ‘blind’ or ‘event-triggered’. In blind BIST diagnosis, the FPGA’s resources are periodically scanned to detect possible faults in the FPGA. Event-triggered BIST diagnosis, on the other hand, is only triggered once an error is detected in the system.

This thesis proposes using TMR as a mechanism for triggering the BIST diagnosis to reduce the impact of the diagnosis time overhead on the system. In fact, by using TMR for the functional modules in the FPGA the impact of BIST diagnosis can be reduced in two ways:

1) The BIST diagnosis is only triggered once a fault is detected by redundancy. This means that the system does not need to be interrupted during normal operation.
2) Once a fault is detected by redundancy, BIST diagnosis only needs to be performed on the region affected by the fault

Figure 5.11 shows the general permanent fault mitigation scheme proposed in this thesis. Initially, a permanent fault affecting a particular module in the FPGA is detected by redundancy. This stage is called the fault isolation stage as an active fault is isolated within the region occupied by the faulty module. The region occupied by a faulty module is defined as the Region Under Test (RUT). Once a fault is isolated within an RUT, the diagnosis process can start by loading the BIST circuits in the RUT to identify the faulty resource. Once the faulty resource is detected, the resource is marked and the functional modules in the FPGA are rearranged by means of bitstream relocation to avoid using the damaged resource.

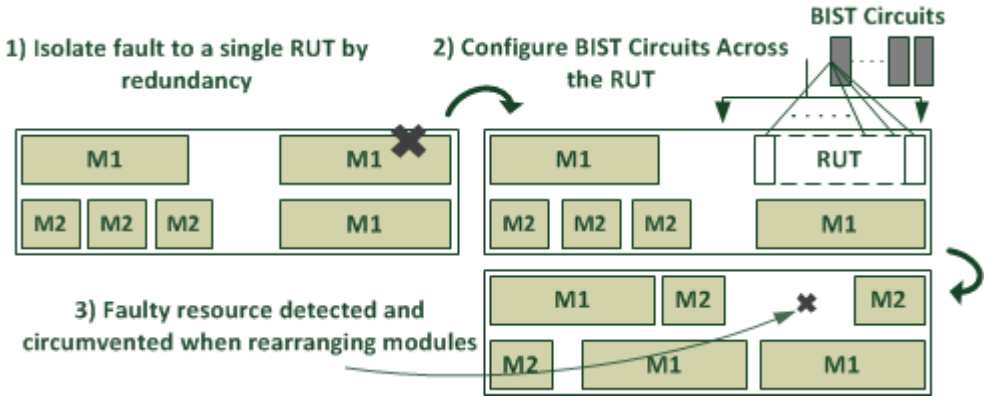



**Figure 5.11 Permanent-fault mitigation**

Most faults in FPGAs are caused by soft faults. It is important to distinguish between soft and hard faults before triggering a BIST diagnosis operation. In addition, some permanent faults are stuck-at faults that cause configuration bits to get stuck at either logic '0' or logic '1'. These faults can be detected with readback scrubbing in a much shorter time than BIST diagnosis. Figure 5.12 shows the proposed fault diagnosis

scheme which takes, into account soft faults, stuck-at faults as well as other types of permanent fault. The first stage of the fault diagnosis starts when a faulty module is detected by redundancy. In the first stage, the module is reconfigured and the erroneous computation is repeated by the reconfigured module. If the error is persistent, the diagnosis enters the second stage. Otherwise, the fault is identified as a soft fault and no further action is taken. In the second stage of the diagnosis scheme, readback scrubbing is performed on the reconfigured module. If a bit-flip appears at this stage, the bit is identified as faulty. If readback scrubbing does not show any faults in the module, this means that the origin of the fault is unidentifiable by readback scrubbing and the diagnosis should enter the BIST diagnosis stage. Faults that cannot be identified by readback scrubbing include faults in LUTs configured as SRL16 and distributed RAM.

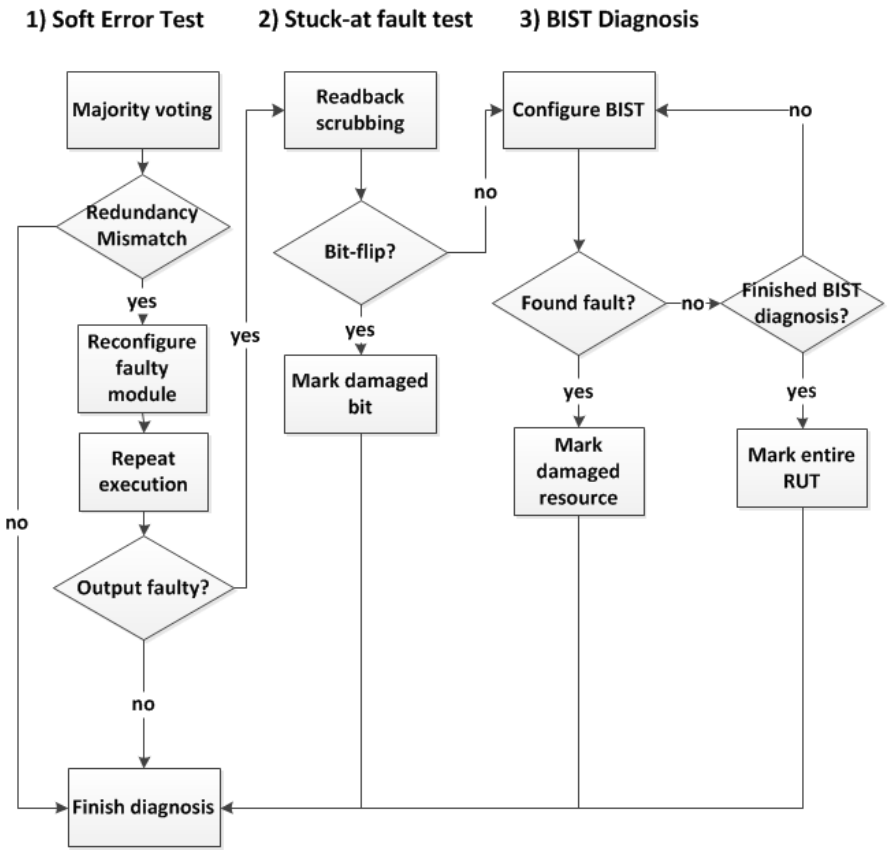



**Figure 5.12 Fault diagnosis**

### 5.3.2 Fast and Scalable BIST Diagnosis

BIST circuits can be used to test a variety of permanent fault types (see Chapter 3). There are two main issues preventing efficient deployment of BIST circuits in reconfigurable systems. The first is the large time overhead which is dictated by the configuration time of the different specialised BIST circuits. A typical online BIST diagnosis system can cause a delay in the order of a second as reported in [141]. The second issue of BIST diagnosis is the large memory required for storing the different BIST circuit's configurations.

To deal with these two issues, this thesis proposes using small relocatable BIST circuits, which can be tiled together to cover any area with any shape in the FPGA. Using relocatable BIST circuits means that only a single partial bitstream is required for each BIST circuit. With the fast relocation method presented in Chapter 4, the BIST diagnosis time overhead can be significantly reduced.

BIST circuits typically have a regular internal structure. Generally, two approaches can be used for tiling BIST circuits to cover a particular RUT. The first approach is based on self-contained BIST circuits, each with its own TPG. The configured circuits are independent of each other. This approach requires each circuit to be enabled separately and also requires the test results to be read from each circuit (see Figure 5.13a). The other approach for tiling the BIST circuits allows the test pattern as well as the test result of each circuit to be propagated through the BIST circuits by having fixed routing inside each BIST circuit. TPGs are only placed in the first CLB column and the test result is read from the last column (see Figure 5.13b).

The benefit of using small relocatable BIST circuits in not limited to the reduced storage memory size. The fact that several identical circuits are tiled together to cover a particular area can greatly reduce the diagnosis time when using the multiple-clone configuration technique (see Chapter 4). With the multiple-clone configuration technique the configuration time can be several times smaller than the conventional configuration technique. This is especially beneficial for BIST diagnosis as several tests may be required to complete the diagnosis process.

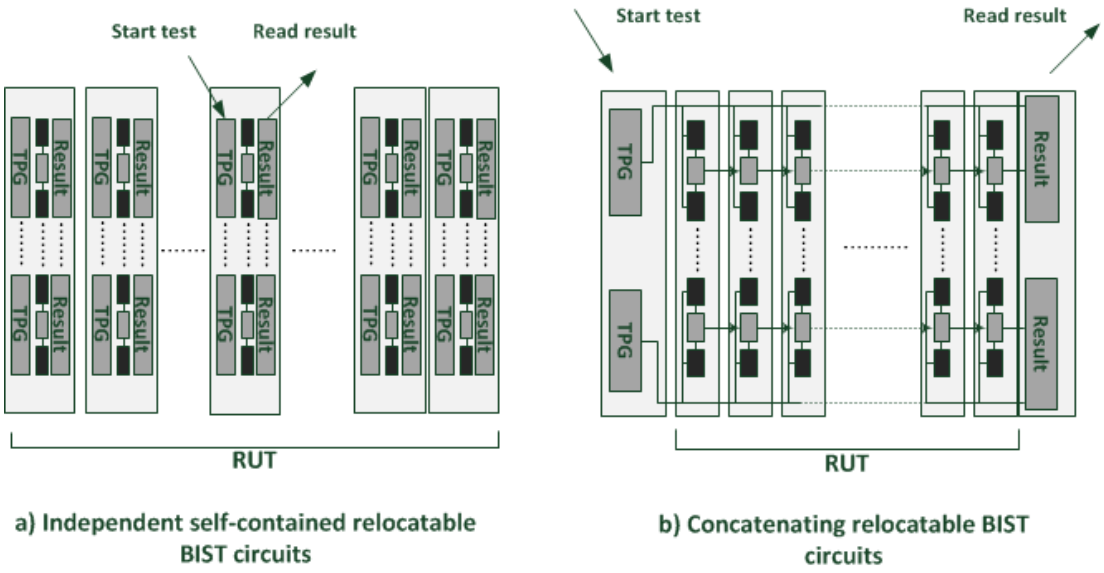



**Figure 5.13 Tiling the relocatable BIST circuits**

There are several types of BIST circuits that are specialised for testing particular resource types in the FPGA. The tiling method can be applicable to any BIST circuit, provided that it has a regular internal structure. LUTs in FPGAs are commonly addressed in BIST diagnosis schemes as these resources are not covered by readback scrubbing when used as shift registers or distributed RAM.

This thesis proposes a single column self-contained BIST circuit which tests the operation of LUTs in a CLB column. In the BIST circuit, two LUTs are connected to a single ORA. The ORA can be implemented using a single 3-input LUT with a flip-flop (see Figure 5.14). The output of the ORA's flip-flop remains at logic '1' when a mismatch between the two ORAs input occurs. This can be achieved by initialising the ORA's LUTs according to the truth table shown in Table 5.5. The TPG is a 4-bit counter connected to the input of the LUTs, which tests all the possible outputs. To account for stuck-at faults, two configurations are required for the tested LUTs where the LUTs in each configuration are initialised with the patterns '101010…' and '010101…', respectively.

The proposed BIST circuit is self-contained with no Bus-Macros for routing to the BIST circuits. This means that controlling the BIST circuits as well as fetching the

test results from the circuits must be done using configuration write/read operations through the ICAP. The 'enable' signals of the TGPs are connected to the output of a dedicated LUT placed in a specific location known for every configuration. The input pins of this LUT are tied to ground pointing to the first address, which is initialised with logic '0'. To enable the TPGs, a readback operation is performed for the frame containing the configuration bits for that LUT and the bit that drives the output of the LUT is set to logic '1'. In the case of multiple BIST circuits aligned to cover a given area, this operation is repeated for each BIST column. When the LUT test for a single configuration is finished, the outputs of the ORA's flip-flops are readback to determine if there was a faulty component. Reading back the current state of flip-flops is possible using the GCAPTURE property in Virtex FPGAs [25]. The GCAPTURE property updates the readback data with the current state of the FPGA's flip-flops by asserting the input signal of the GCAPTURE primitive. In Virtex FPGAs, the current states of flip-flops within a CLB column are stored in a single frame. In the case of a Virtex-4 FPGA, this frame is the 20$^{th}$ minor frame.

**Table 5.5 Truth table for the ORA's 3-input LUT**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **INPUT1** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| **INPUT2** | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| **INPUT3** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| **OUTPUT** | **0** | **1** | **1** | **0** | **1** | **1** | **1** | **1** |

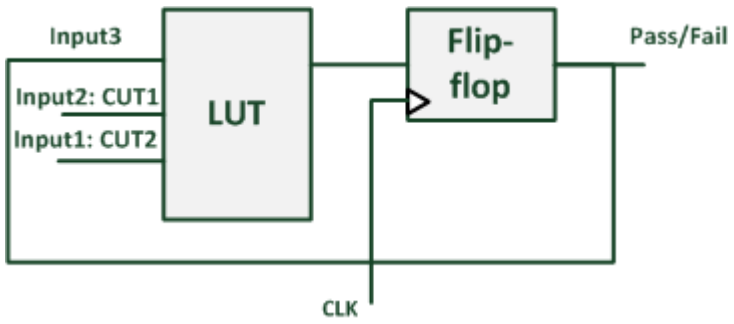



**Figure 5.14 ORA implemented with a 3-input LUT and a flip-flop [125]**

Because some of the LUTs are used for ORAs and TPGs in the BIST circuit, more than one configuration is needed to test all the LUTs within the CLB column. To test a Virtex-4 CLB column, in each configuration half of the LUTs are used as CUTs and ORAs where the other half contains the TPG's logic and the 'enable' LUTs. In the first BIST configuration, the top 32 SLICEL and SLICEM LUTs are used for CUTs and ORAs where all 32 SLICEM LUTs are configured as CUTs and 16 out of the 32 SLICEL LUTs are configured as ORAs (see Figure 5.15). In the second configuration the same arrangement of LUTs and ORAs are configured but with different initialisation patterns to test for stuck-at faults. In the third and fourth configurations, the CUTs are placed in SLICEL LUTs and the ORAs are placed in SLICEM LUTs, again with different CUT initialisation patterns. After four configurations, the 64 LUTS on the top of the CLB column are tested. Four more configurations are required to test the operation of the bottom 64 LUTs, giving a total of eight configurations to complete the BIST test for the 4-input LUTs in the CLB column. Each configuration requires storing a single column partial bitstream that contains the configuration of 22 frames.

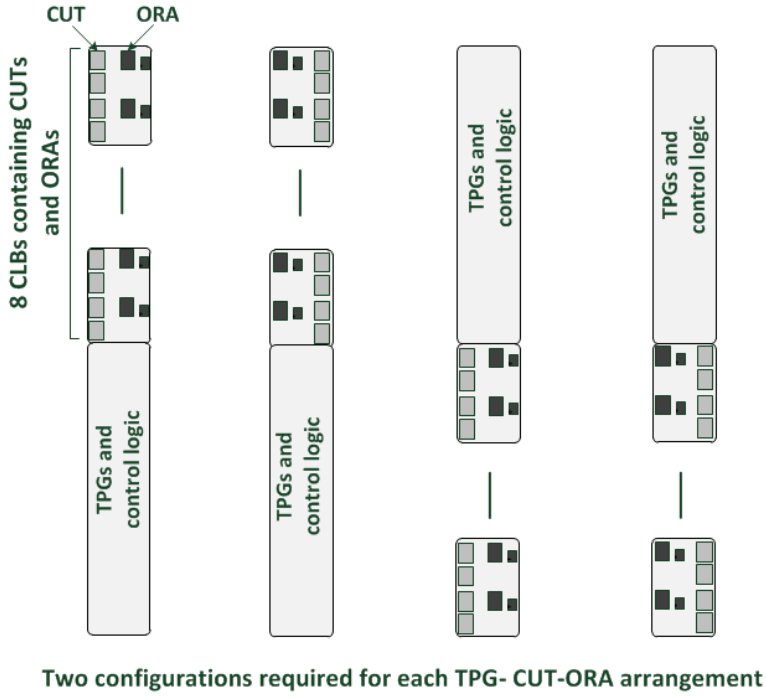



**Figure 5.15 CUT, ORA and TPG arrangement in BIST circuits**

### 5.3.3 BIST Diagnosis Evaluation

As mentioned earlier, BIST diagnosis time overhead is dictated by the configuration time for the different BIST configurations. The control and result fetching times are small: one read-modify-write operation is required to assert each 'enable' LUT and one readback operation is required to fetch the test results from each CLB column.

With the offset-based FAR modification technique (see Chapter 4), the relocation process of a BIST partial bitstream can be performed online with minimal delay. In addition, the multiple-clone configuration feature presented in Chapter 4 can be applied for the BIST circuits configuration where multiple clones of the same BIST circuit are configured using a single compressed partial bitstream generated online instead of configuring each BIST instance individually. This will result in a significant reduction in the BIST configuration time overhead, especially when a large number of configurations are required.

Figure 5.16 shows the configuration time in the LUT-BIST diagnosis with and without the multiple-clone configuration technique. It can be seen from Figure 5.16 that the configuration time for a typical RUT size can be more than three times smaller with BIST cloning.

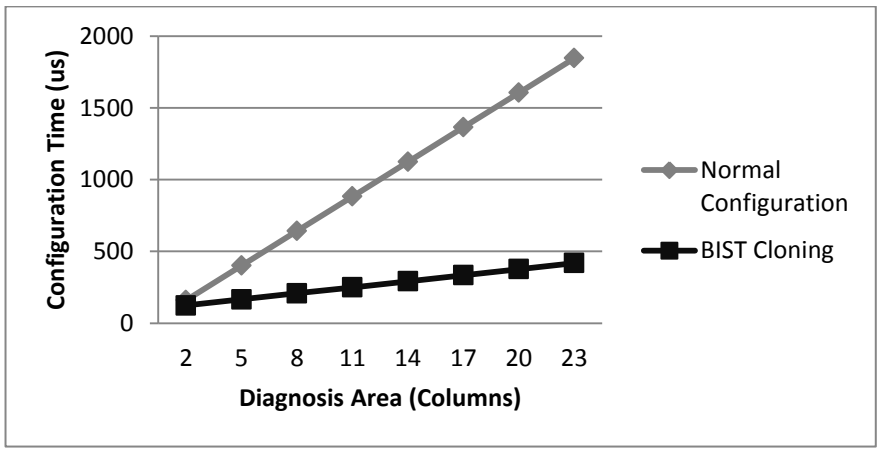



**Figure 5.16 Configuration time in BIST diagnosis**

It can be argued that the same configuration acceleration can be achieved by offline bitstream compression where the MFW feature is used to generate several BIST configurations for different areas and shapes [125]. While this is true, this will still require a different set of BIST configurations for every area shape and size leading to a large battery of BIST circuits. Figure 5.17 shows the memory required for storing the configurations of different BIST circuits with different sizes. It is noted that with the BIST tiling technique the memory savings can be in the range of Mbytes in systems where multiple areas with different shapes are diagnosed. One example of such a system is an ROS that executes HTs with a variety of sizes and shapes. Rather than storing a set of BIST configurations for each HT, a single set of BIST configurations can be used to diagnose the area occupied by any HT.

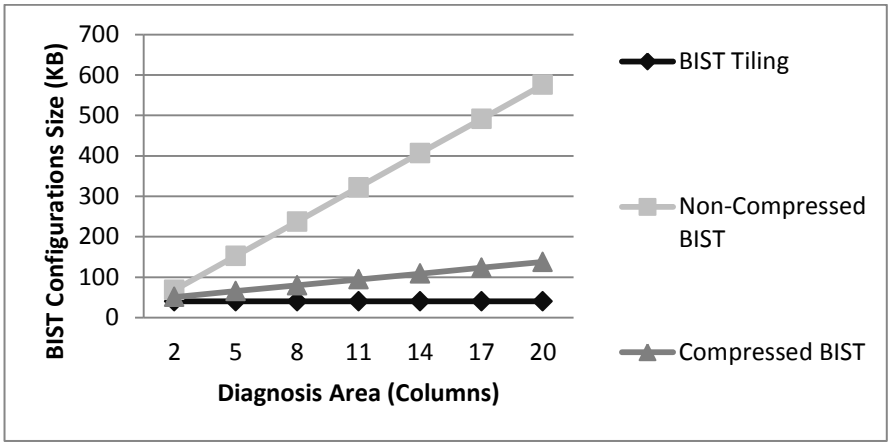



**Figure 5.17 Storage memory required for BIST configurations**

## 5.4 The Reliable Reconfigurable Real-Time Operating System

Designing a comprehensive FT system for a particular application can be difficult and time consuming, especially when the system is required to mitigate both transient and permanent faults. Dealing with permanent faults in particular is very complex as it requires modifying the original configuration of the system. When offline configuration upgrade is not feasible, this requires applying online self-

healing techniques, which usually deploy bitstream relocation. FT applications should be designed within a generic platform with proven 'fault-resilience'. Ideally, the platform should enable designers to write their applications without dealing with the complexity of fault detection and recovery. Indeed, writing applications over a reliable ROS will allow designers to easily modify and upgrade their FT applications, especially when high-level programming is supported.

The R3TOS is a computational platform specifically designed for writing applications that require both high-performance and reliability in FPGAs ([17] and [16]). R3TOS gives support for reliable execution of real-time HTs by deploying all the fault detection and recovery methods presented earlier in this chapter. The ultimate goal of R3TOS is to make the fault handling transparent to designers who are only required to follow a set of rules for designing the hardware modules deployed in the target application. The software layer in the target application can assign specific tasks to these hardware modules, which are configured online by means of DPR. Task assigned to hardware modules (referred to as 'HTs'), are managed by the R3TOS microkernel, which heavily utilises the ICM for configuration of HTs as well as performing the fault detection and recovery operations. The R3TOS microkernel also contains a scheduler to determine the order of HT execution and a fault-aware allocator, which allocate the scheduled HTs on the available resources of the FPGA, thereby avoiding any damaged resource ([142] and [143]) (see Figure 5.18).

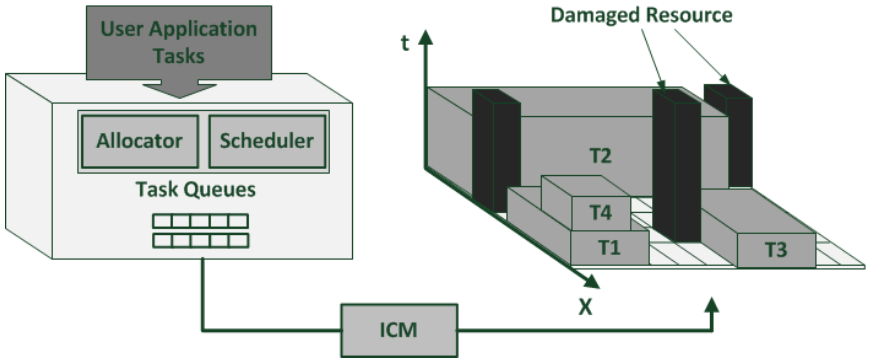



**Figure 5.18 R3TOS [17]**

### 5.4.1 R3TOS Architecture

The R3TOS system consists of two main parts. The first part contains the static control components, which include the main CPU running the target application's software, and the ICM, which controls the configuration operations in the system. The second part of R3TOS is the reconfigurable region used for the configuration of the relocatable hardware modules of the target application. The system is floor-planned so that the static part is constrained within a dedicated region in the FPGA. The local routes of the static part are constrained as much as possible to the area within the static region. In addition, the external IOs of the system are limited to IOs located in the static region or at the boundaries of the static region. This imposes that the static region is placed in one of the corners of the FPGA chip. This placement constraint leaves the remainder of the FPGA chip almost empty and free of static routes. The reconfigurable part is selected to be within this empty region so that relocatable hardware modules can be placed freely at run-time.

The regional clock buffers of the FPGA are all instantiated when floor-planning the system. All the regional clock buffers are fed by the same global clock buffer, which is connected to the external clock source through one of the systems IOs. Figure 5.19 shows a simplified diagram of the R3TOS architecture.

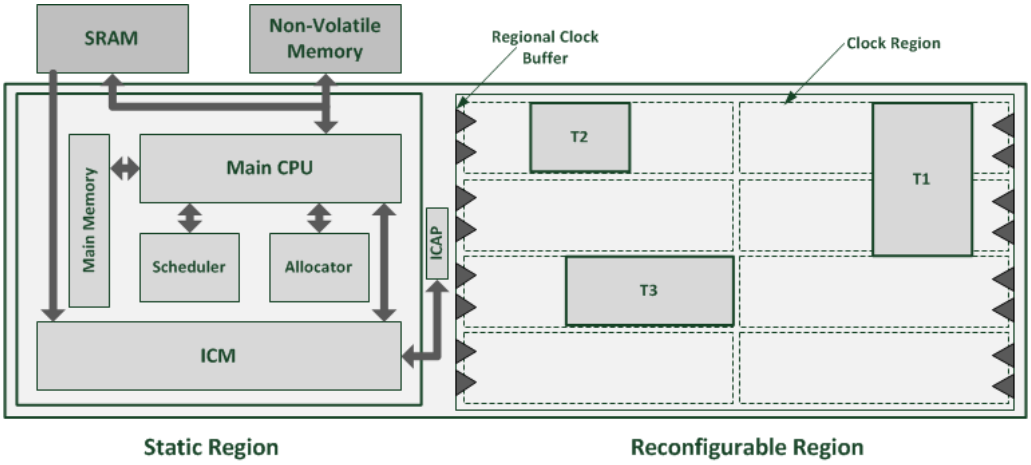



**Figure 5.19 Simplified R3TOS architecture**

### 5.4.2 Online Routing

Most of the systems that support bitstream relocation depend on a fixed infrastructure of bus-macros for communication between the static logic and the reconfigurable modules (see Chapter 2). Using fixed bus-macros limits the total number of feasible locations for each relocatable module, making such systems not suitable for an ROS implementation, especially when permanent faults are taken into account in the ROS operation. Online routing can increase the total number of feasible locations for the relocatable modules. However, the time overhead for physically rerouting the system is tremendous and requires a deep knowledge of the FPGA's routing resources.

On-chip communication in R3TOS is based on a virtual bus over the configuration layer of the FPGA [142]. In other words, the physical routes in the FPGA are not used for connecting the relocatable modules to each other and to the static components in the system. Instead, ICAP read and write operations are used to transfer data to/from each relocatable module (see Figure 5.20). Using the configuration layer of the FPGA for on-chip communication eliminates the need for a fixed routing structure and greatly increases the flexibility of module relocation. The R3TOS communication scheme requires each relocatable module to have an Input Data Buffer (IDB) and an Output Data Buffer (ODB). The data buffers can be made out of LUT distributed RAM or made out of BRAMs. BRAMs are preferable for larger buffers as distributed RAM can consume a lot of the FPGA's resources. Data can be exchanged between buffers by reading the configuration of the 'source' buffer and then copying this configuration to the 'destination' buffer. This communication method dictates that data is transferred between buffers in separate segments rather than a continuous stream. Due to the limited throughput of the ICAP, the ICM must efficiently manage the data transfer tasks and operate at the maximum speed possible in order not to degrade the communication bandwidth [144].

When designing an R3TOS relocatable module, the resources of the module are clocked by a default regional clock buffer. The default regional clock buffer is the

middle clock buffer within the height of the module. As the entire regional clock buffers are instantiated with the same configuration in R3TOS, the fixed clock distribution will allow modules to be relocated freely between the clock regions of the FPGA (see Figure 5.21). R3TOS also provides a mechanism for controlling the clock signal going through relocatable modules by modifying the configuration of the regional clock buffers via the ICAP. Any regional clock can be enabled or disabled as desired by enabling/disabling the PIP connecting the clock net to the clock region. In addition, the clock frequency of each regional clock net can also be modified online by changing the configuration of the regional buffers. In particular, the configuration of the regional clock buffer contains a 'clock divide' parameter, which divides the clock by an integer. To perform these clock modifications, some knowledge of the configuration bits of the FPGA's clocking resources is required. The functionality and location of the regional clock buffers can be found by performing simple analysis on the FPGA's bitstream [27].

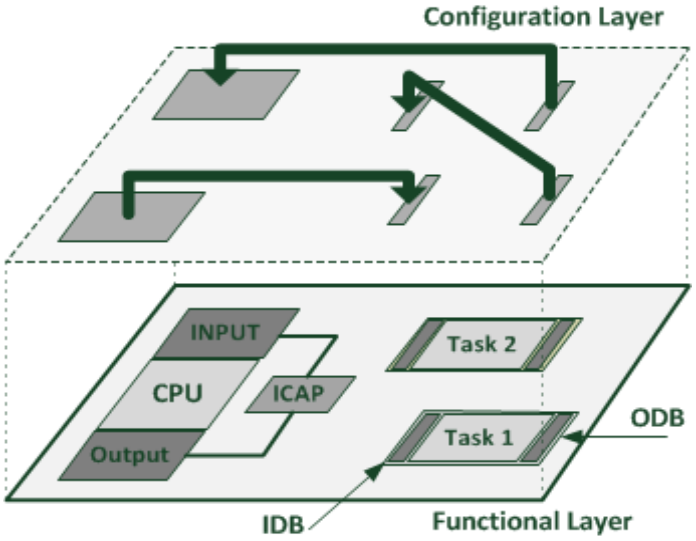



**Figure 5.20 ICAP-based data transfer [144]**

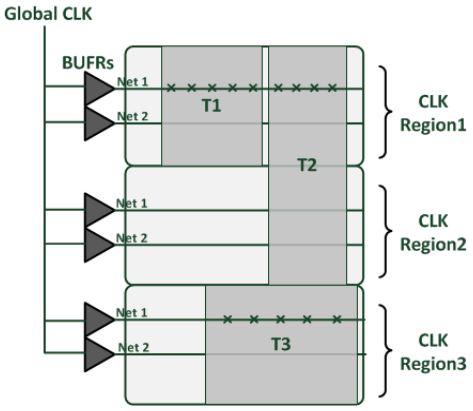



**Figure 5.21 Fixed clock distribution [27]**

## 5.4.3 HT Management

The relocatable modules in R3TOS are fully isolated. They are specifically designed so that no physical routes are required for data transfer and control signals (e.g. enable, ready signals). The R3TOS uses binary semaphores to control the operation of the relocatable modules. These semaphores are control bits used for managing the relocatable cores using the ICAP. These control bits can be either stored in dedicated LUTs or can be embedded inside the data buffers of the relocatable module (see Figure 5.22). The relocatable module should also contain a small control FSM that controls the operation of the module and the internal data flow from/to buffers.

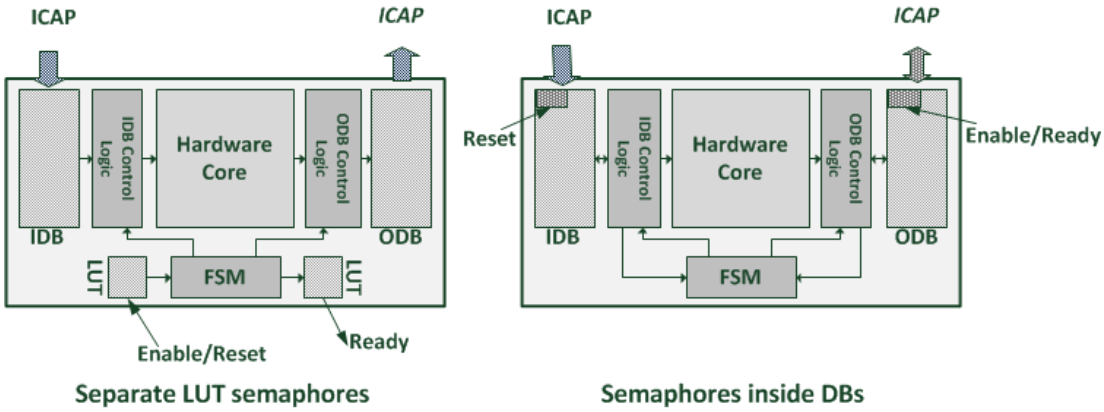



**Figure 5.22 The relocatable module architecture**

It is important that any memory element inside the relocatable module is not accessed by the ICAP and the FSM's internal logic at the same time as this can corrupt its content. In the case of LUT semaphores, it is also important not to corrupt the content of other LUTs that are placed in the same column as the semaphores when modifying them. Protecting the content of the LUTs can be accomplished by disabling the active clock in a clock region when any semaphore within the region is accessed by the ICAP. Disabling a regional clock buffer will briefly freeze the operation of the tasks operating inside the region. Figure 5.23 shows the steps required to safely transfer data from/to the relocatable modules when using BRAMs for data buffers.

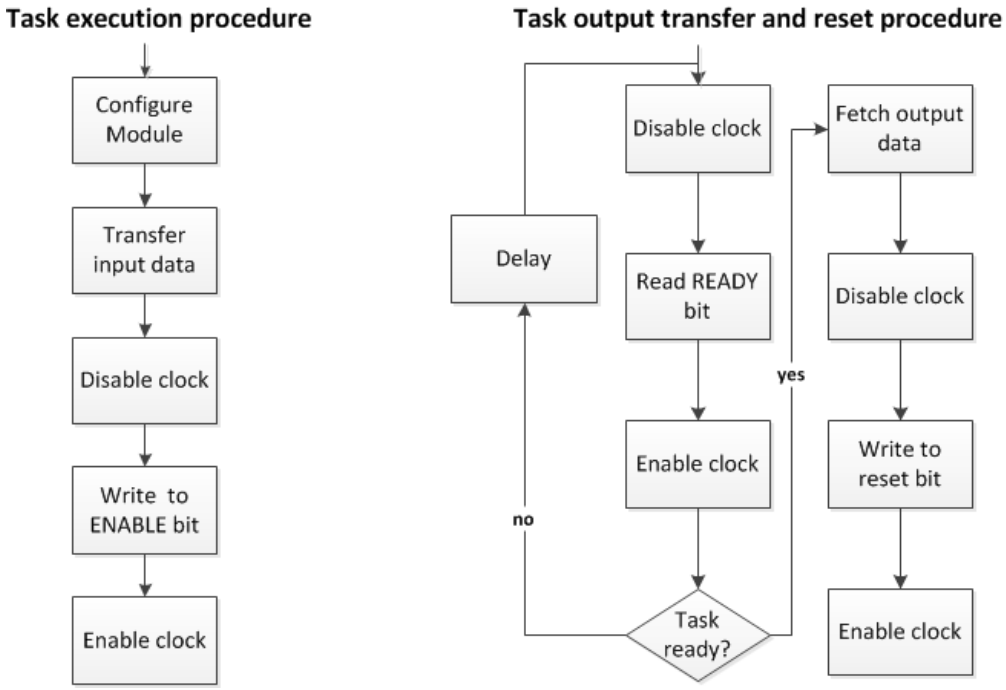



**Figure 5.23 HT execution management**

The FSM inside a relocatable module coordinates the dataflow inside the module and makes sure that the semaphores and the buffers are not accessed by the ICAP when they are internally active. The operation of the FSM differs depending on the type of semaphores chosen for the module. For semaphores embedded into the data buffers the reset bit is placed in a known location in the IDB, whereas both the enable and

ready semaphores share the same bit inside the ODB (i.e. the ICAP asserts the bit to enable the module and the FSM de-asserts the bit when it finishes execution). It is noted that, for modules containing semaphores inside the buffers, the IDB and the ODB should be placed in different BRAM columns. For modules containing LUT semaphores, the enable and reset semaphores share the same bit in the LUT (i.e. ICAP asserts the bit to enable the module and de-asserts the bit to reset the module). The ready semaphore, on the other hand, is placed in a dedicated LUT. Figure 5.24 shows the FSM operation when using the two types of semaphore.

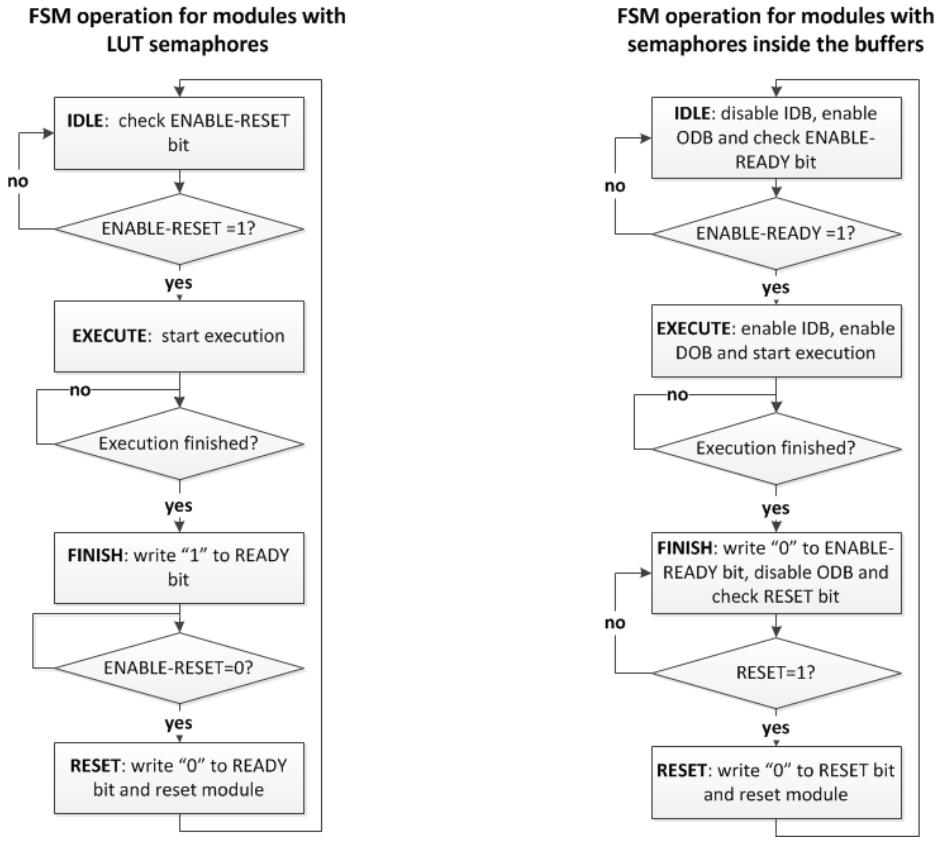



**Figure 5.24 Relocatable module's FSM operation**

## 5.5 Chapter Conclusion

This chapter presented different fault detection and recovery methods that can be applied to the ICM to realise fault-tolerant self-healing systems in FPGAs. The self-healing capabilities allow for both transient and permanent fault detection and mitigation.

The soft error mitigation is based on configuration memory scrubbing. Different scrubbing techniques were evaluated and a scrubbing scheme that combines both fast external and internal readback scrubbing was proposed to allow for better fault coverage and to reduce the total number of single points of failure. In addition, the ICM was designed with self-healing capabilities. The design of the proposed ICM has a reduced resource utilisation compared to conventional TMR designs. In the proposed ICM, TMR is only applied to a small portion of the ICM, which is capable of recovering the rest of the circuit once an error is detected by monitoring the ICM's operation and the configuration CRC.

Permanent error mitigation is based on modular relocation where any faulty module in the system is relocated to a new location on the FPGA at run-time. The R3TOS scheme model was presented. This relocation scheme allows for fully isolated modules to be freely relocated between clock regions and, hence, allow for better permanent fault mitigation.

This chapter also presented a novel online BIST diagnosis technique aimed at detecting emerging permanent defects in the FPGA. The proposed diagnostic technique exploits the multiple-clone configuration technique to 'clone' (i.e. replicate) a single basic BIST circuit along arbitrarily sized and shaped areas on the FPGA without incurring large time overheads. Hence, the proposed technique allows for the creation of run-time on-demand tailored BIST circuits to satisfy any diagnosis requirements that may arise. Moreover, the proposed solution allows for saving memory in the system as it only requires storing the configurations of a single basic BIST circuit.

Finally, this chapter presented the R3TOS computing platform, which integrates all the presented FT features in a single ROS. A novel HT management system that manages the execution and data transfer of tasks is also presented. This system utilises the ICM to enable flexible relocation of fully isolated tasks, making the system capable of efficiently handling permeant faults in the reconfigurable resources.

# Chapter 6 : An R3TOS-based Reliable and Secure Encryption Engine

FPGAs have become a popular target for implementing cryptographic block ciphers. An optimised design of a block cipher in an FPGA can combine the flexibility and low cost of software solutions with high throughputs that are comparable to custom ASIC designs. There is a huge amount of research focused on the implementation of a wide range of popular cryptographic functions in FPGAs. In [145], Elbirt et al. present several implementations of the Advanced Encryption Standard (AES) and show how the AES can be optimised for performance in FPGAs to be at least an order of magnitude faster than most software implementations. In [146], Good et al. discus how the AES can be optimised to reduce the resource utilisation and power consumption in small FPGAs for low power mobile applications. FPGAs has also been a target for the implementation of high-performance stream ciphers [147]. In addition the FPGA market is witnessing a rising number of companies providing third party cryptography IPs for FPGAs (e.g. [148] and [149]).

The run-time reconfiguration capabilities of FPGAs have opened the door for some interesting cryptographic applications. There are several ways in which DPR can be harnessed in cryptographic applications (adapted from [150]):

1) Algorithm switch: there are many standard cryptographic functions that can be implemented in FPGA. By having a battery of bitstreams, each for a specific cryptographic function, the range of standards supported by the same device can be extended.
2) Algorithm upgrade: in FPGA systems, the device configuration can be updated even remotely. This allows for a longer life-time as the system can be upgraded to emerging security requirements. From a cryptographic point of view, this leads

to a more secure system as emergency measures can be taken to change the implemented algorithm if it has been broken.

3) Resource efficiency: many cryptographic applications are based on hybrid protocols that require more than one cryptographic algorithm. For example, secure data transmission requires one algorithm to establish a secure data transmission session and a second algorithm for data encryption. Since the algorithms are not used simultaneously, DPR can be used to allow the use of the same hardware resources for the two algorithms and, hence, achieve a better resource efficiency.

## 6.1 Background on FPGA Security

The protection of IP cores is one of the main concerns of FPGA manufactures and companies providing third-party support for FPGAs. There are several motives for attackers to clone IPs implemented on FPGAs. For example, an attacker could make financial gains through unlicensed deployment of IPs. If an attacker has the knowledge required to reverse-engineer the FPGA bitstream, trade secrets can be revealed posing more serious damage to the IP owner. On the other hand, an attacker might be interested in revealing secret information in the data handled by the FPGA during its operation. As FPGAs have become a popular platform for implementing cryptographic functions for various applications, more research is gearing toward the issue of data security in FPGAs.

### 6.1.1 Basic Security Features in Commercial SRAM FPGAs

In SRAM FPGAs, a bitstream is required to be stored in a non-volatile memory for the configuration of the FPGA. This makes any IPs vulnerable if the proper security measures are not taken to protect the bitstream. FPGA vendors provide different solutions to prevent IP cloning. The most common method of IP protection against cloning is to encrypt the IP bitstream and store it in a secure non-volatile memory. A key stored inside the FPGA is used by an internal decryption circuit to decrypt the bitstream before the configuration process. The key can be stored in a dedicated non-

volatile memory or a dedicated volatile memory backed by an external battery ([151] and [152]).

Cipher text attacks can be used to alter the functionality of a protected bitstream by tampering the bitstream. There are several solutions available to address this issue. One solution is the use of parity bits such as CRC-32 to check the integrity of the bitstream during configuration [153]. Other solutions are based on hand-shaking protocols and token exchange between the FPGA and the authorised configuring device [154].

As mentioned earlier, DPR is an attractive feature for many cryptographic applications; however, it can be a security hole as an attacker can use this capability to insert hardware Trojans. Modern FPGAs allow for disabling readback and DPR from any external configuration port. In fact, in Xilinx FPGAs, external access to the FPGA's configuration memory is automatically disabled when an encrypted bitstream is loaded into the device. In recent Virtex FPGAs, readback and DPR are only possible using the ICAP when bitstream encryption is used. This is because a configuration controller implemented in the FPGA and configured using an encrypted partial bitstream is considered as a trusted channel for DPR.

### 6.1.2 Side Channel Attacks: Vulnerabilities and Countermeasures

Power analysis attacks were first introduced by Kocher in 1998 as a distinct class of side channel attacks [155]. They are based on analysing the power consumption measurements of tamper resistant devices to find secret keys embedded in these devices. Initially, small devices such as smart cards and simple processors were the target of such attacks. In recent years, the advances in power analysis techniques have extended the range of valuable devices. Several research studies were focused on the threat of power analysis attacks as a method for retrieving keys embedded in FPGAs ([156] and [157]). Practical successful attacks to break the bitstream encryption of some FPGA families were also reported ([158] and [159]).

The threat of power analysis attacks on FPGAs has led to the development of several countermeasures. One example is the system proposed in [160], where an internal circuitry is used to monitor the power supply voltage to detect possible insertion of power measurement circuits onto the device's power rail. The Xilinx commercial Security Monitor IP can also be used in a similar manner to detect suspicious variations in temperature and voltage after configuration [161]. Other power analysis attack countermeasures are based on internal manipulation of the power consumption. For example, the system in [162] contains an on-chip 'power consumer' circuitry, which is used to keep the power consumption of the system constant to reduce the possibility of leaking information through power consumption. Deliberate power consumption can also be used to insert noise in power measurements to increase the difficulty of power analysis attacks [163].

Fault injection attacks are another type of attack that could potentially pose a risk to FPGA security. Fault injection attacks are based on analysing leaked secret information of cryptographic functions caused by malfunction in their hardware when certain faults are injected into the system. This kind of attack was first introduced in [164] wherein the authors demonstrated how to break a public key algorithm such as the RSA by exploiting faults in the system. More advanced differential fault analysis attacks that could potentially be applied against all known symmetric cryptographic functions were later introduced in [165]. There are several ways for attackers to inject faults in electronic circuits such as the use of infrared laser and electromagnetic radiation ([166] and [167]). The authors in [168] have classified the hardware countermeasures against fault injection attacks into two categories: passive countermeasures and active countermeasures.

Passive countermeasures aim at increasing the difficulty of inserting faults into the protected device. For example, applying a metal shield that covers a protected chip makes fault injection through electromagnetic radiation and laser beams more difficult as the shield needs to be removed in order for the attack to succeed.

Active countermeasures are based on taking certain actions when fault attacks are detected. Integrating light detectors, voltage detectors and frequency detectors are

common techniques to detect changes in light gradient, voltage and clock frequency. Active shields can also be used against fault attacks. Active shields are metal mesh layers that cover the entire chip and have data continuously passing through them [168]. Attacks can be detected when a discontinuation of the data passing through the shield occurs due to tampering with the device. Despite the aforementioned countermeasures being effective in detecting fault attacks, they require the integration of special components into the FPGA chip. The most common active countermeasures against fault attacks are based on classic fault detection and mitigation techniques such as modular redundancy and parity checking [169]. In fact, Xilinx has already started advertising its FT solutions as countermeasures to fault attacks [170].

## 6.2 Overview of the Encryption Engine

The security of electronic devices is related highly to their reliability. Not only faults deliberately injected into electronic devices can pose a security threat, but also naturally occurring random faults can potentially lead to the leaking of secret information. This issue is particularly important in space applications in which electronic devices are operating under high levels of radiation. In space application, the ability of deploying cryptographic functions in reliable reconfigurable hardware is very attractive and beneficial. While previously proposed systems have already demonstrated how DPR can be used for implementing a wide range of cryptographic standards within the limited FPGA resources (e.g. [171] and [172]), this section of the thesis demonstrates how critical encryption tasks can be implemented using the R3TOS with special emphasis on the reliability of task execution. Using R3TOS, the FPGA chip can be used as a server of a wide range of cryptographic tasks that can be executed reliably and securely. The multi-tasking capabilities of R3TOS can be used to serve cryptographic functions of multiple users or multiple applications running at the same time. Figure 6.1 shows the proposed adaptation of R3TOS as a cryptography server.

In Chapter 5 the reliability capabilities of R3TOS were explained in detail. In the context of the proposed server solution, these capabilities are deployed to meet several reliability criteria:

1) Transient faults could affect the system in several ways. While faults in some tasks can cause errors in the functionality of the system, other faults can potentially lead to the leaking of secret information. Tasks of a cryptographic nature are defined as critical tasks. These tasks are always performed in the reconfigurable region using multiple redundant hardware cores.
2) The system always keeps track of the permanently damaged resources in the reconfiguration area. When allocating tasks, these resources are circumvented. This capability is particularly important in long space missions to allow the system to adapt to emerging permanent faults.
3) An FT version of the ICM is used to reduce the probability of faults occurring during the configuration of the cryptographic cores. The rest of the R3TOS components are protected by means of configuration memory scrubbing.

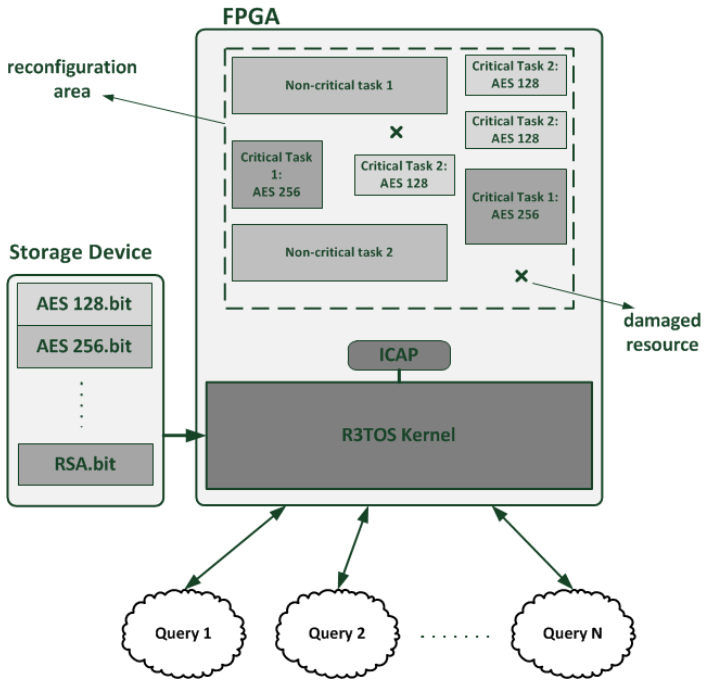



**Figure 6.1 R3TOS cryptography server**

### 6.2.1 The Relocatable Cryptographic Core

The cryptographic cores used in the proposed system follow the R3TOS relocation architecture. Each core is designed to be self-contained with all routes constrained within the area occupied by the core. Communication and data transfer are accomplished via read/write operations through the configuration layer of the FPGA. Figure 6.2 shows the generic architecture of the relocatable cryptographic core.

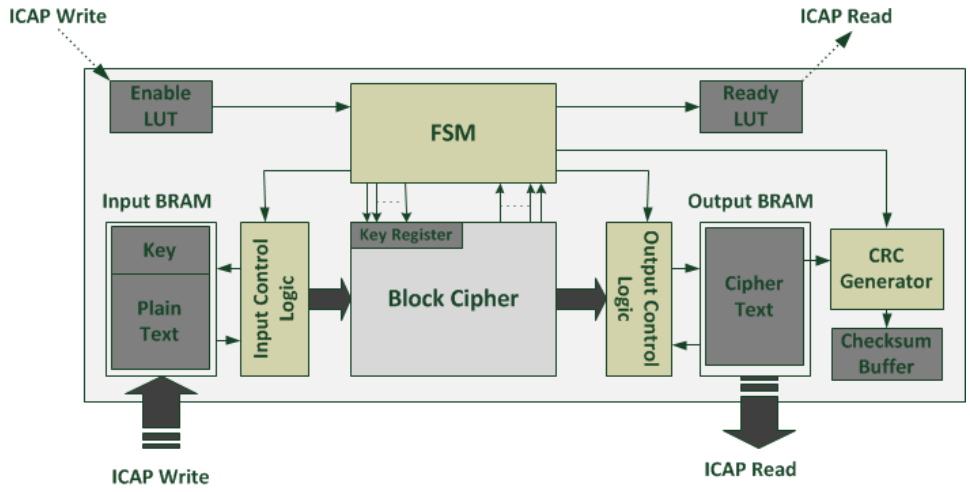



**Figure 6.2 Generic architecture of the relocatable block cipher**

A relocatable cryptographic core contains a block cipher, an FSM, a CRC-generator and some memory components along with their control logic. The memory components are used for communication with the R3TOS kernel via the ICAP. Input BRAM blocks are used for transferring the key and plain text to the core, and output BRAM blocks are used to transfer the cipher text when the task has completed. The FSM triggers the operation of the core when an enable signal is written to a dedicated LUT by the ICAP. When the cipher block finishes its operation, the CRC generator generates a checksum for the cipher text and stores it in a dedicated LUT buffer. Finally, the FSM writes a 'ready' flag in another LUT to indicate that the output of the core is ready for collection.

### 6.2.2 Online Placement of Heterogeneous Cores

There is a large amount of research related to allocating tasks to the available resources in reconfigurable hardware (see Chapter 3). Task allocation is usually approached as a 2-D packing problem where the task's cores are continuously placed in a large area of identical computing resources. It is very difficult to practically apply such placement algorithms to FPGAs as they do not consist of uniform areas of identical resources.

In R3TOS, a 'sandbox' of CLB resources is designated for the placement of the task's cores. The sandbox is defined as the largest area in the FPGA consisting of a uniform arrangement of CLB columns. By limiting the relocatable hardware cores in the systems to those using CLB resources only, the 2-D packing algorithms can be practically applied to FPGAs [143].

Cryptographic cores require a relatively large amount of data to be transferred into the core. Although input/output buffers can be constructed using LUTs, BRAM buffers must be used for storing the plain text and the cipher text due to their large sizes. This makes the 2-D packing algorithms not practical for this application. Generally, two methods can be used to enable online placement of heterogonous cores. In the two methods a matrix stored in memory can be used to map to the FPGA resources. This matrix represents the state of the FPGA resources. For example, used resources can be mapped with logic '1', while available resources can be mapped with logic '0'.

The first method is based on pre-computing all the possible locations of each relocatable core. Location parameters are stored in memory for each relocatable core. When a task is scheduled for configuration, the placer scans the resource matrix and checks which of the possible locations is available before deciding the optimal location of the task. This approach is fast, especially when used with a First-Fit (FF) algorithm as the placer only scans specific locations within the FPGA resource matrix. This approach, however, is not suitable for systems deploying a large number

of relocatable cores due to the memory overhead required to store the location parameters for the cores.

The second approach is based on storing parameters for the layout and dimensions of each relocatable core. When a task is scheduled for configuration, the placer creates a 'window' containing the exact resource layout as the required core and starts moving this window across the resource matrix until a location that fits this window is found [173]. The memory overhead for this approach is very small as only a few parameters for each bitstream are required; however, the resource matrix scanning time overhead is much larger compared to the first approach.

In this thesis, a hybrid algorithm for the resource matrix search is proposed whereby only pre-computed horizontal location offsets are stored in memory. Similar to the aforementioned two placement methods, a resource matrix representing the state of the FPGA resources is stored in memory. Since the smallest partial bitstream has the size of a single column, the matrix is created so that each column in the FPGA is represented with an element in the matrix. A single bit is used for each matrix element where logic '0' is used to represent an 'available' column and logic '1' is used to represent a 'used' column. The initial matrix used after power-up of the device will only contain logic '1' elements for the columns occupied by the static components in the system. To protect the content of this matrix from faults, the matrix can be stored in a special ECC-BRAM, which utilises parity bits for fault correction and can be automatically generated using the Xilinx tools. Figure 6.3 shows an example resource matrix used to map a system implemented in FPGA.

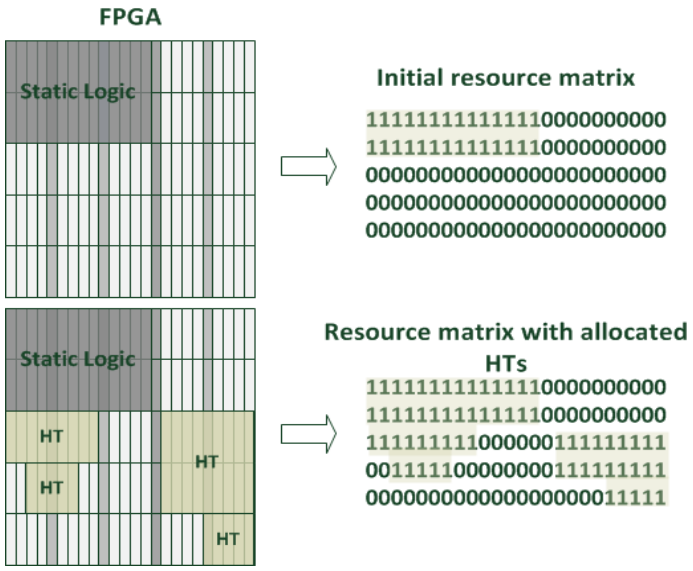



**Figure 6.3 Mapping FPGA resources into a resource matrix**

The resource matrix only represents the state of the columns during the operation of the system; it does not contain any information about the type of these columns. To make placing cores with heterogeneous resource types feasible, pre-computed horizontal offset groups are created and stored in memory. An offset group contains several horizontal offsets, which results in the correct placement of a relocatable core when added to the original horizontal location of the core's bitstream. There are four parameters required to be stored in a relocatable bitstream header to be compatible with the proposed placement scheme. These parameters are the original bitstream location, the offset group ID, the width of the relocatable core and the height of the relocatable core. The original bitstream location parameter contains the horizontal offset at which the bitstream has been generated, whereas the offset group ID parameter contains a number indicating which offset group is compatible with the bitstream. The other two parameters indicate the number of columns the core occupies horizontally and the number of rows the core occupies vertically. Figure 6.4 shows all the offset groups required for the placement of a bitstream consisting of CLB and BRAM resources in a Virtex-4 FX60 FPGA.

As the offset groups contain horizontal offsets from the original locations of the relocatable cores, many cores will share the same offset groups even if they have different resource layouts. From Figure 6.4, we can see that only three offset groups are required for the placement of bitstreams with CLB and BRAM resources in the largest Virtex-4 device. Figures 6.5 shows all the possible horizontal layouts of the relocatable bitstreams for each offset group.

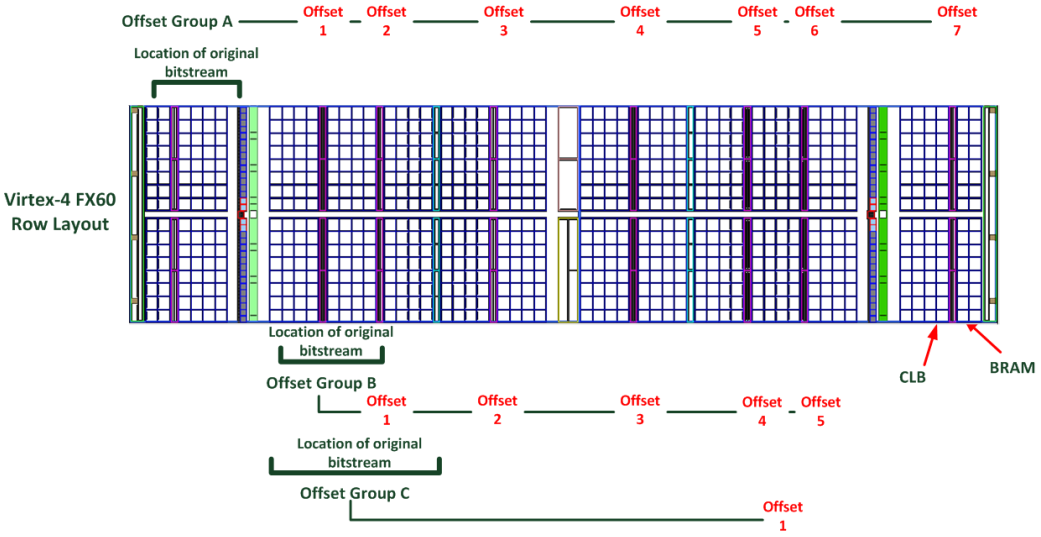



**Figure 6.4 Offset groups for relocatable bitstream consisting of CLB and BRAM resources in a Virtex-4 FX60 FPGA**

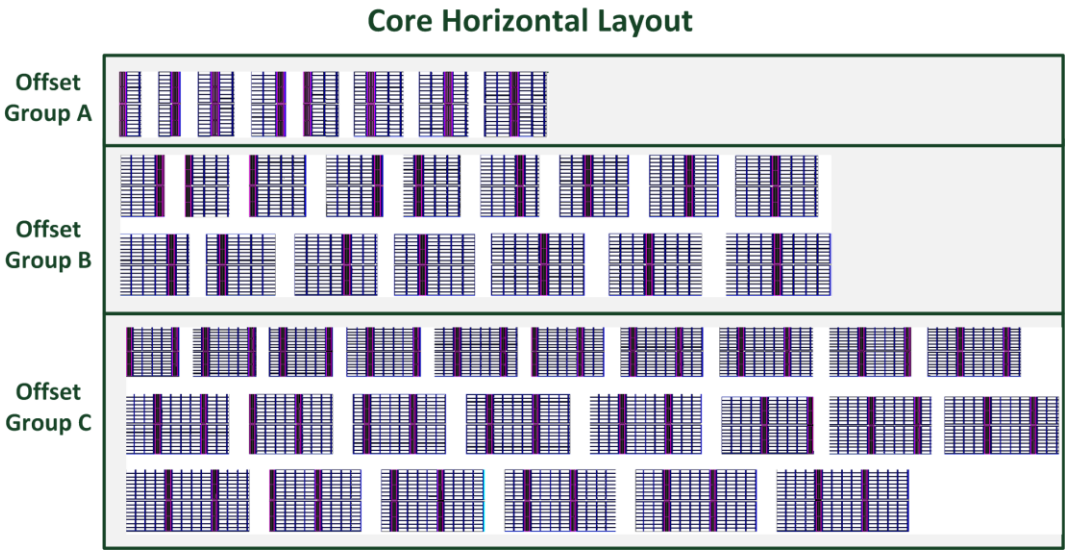



**Figure 6.5 Core horizontal layout's compatibility with offset groups**

The placement algorithm does not need to know the layout of a relocatable core in order to find feasible locations for a core at run-time. The possible horizontal locations of any relocatable core are already computed and stored in the offset group compatible with the core. When a core is scheduled for configuration, the placer only needs to scan the resource matrix vertically at the pre-computed horizontal offsets for that core. Algorithm 6.1 is an FF vertical scan algorithm that returns vertical location parameters for the first location found to fit a relocatable core. For convenient explanation of the algorithm, a two-dimensional array representing the resource matrix and a structure for the location parameters are defined as follows:

```
int matrix [matrix_height] [matrix_width];    //resource matrix: this matrix is initialised with the
                                                initial state of the FPGA

struct location{                 // location parameters
   int Y_offset;                 // vertical offset
   int X_offset;                 // horizontal offset
   int flag;                     // a flag indicating that a location has been found during the
                                    vertical scan
```

The placement of a relocatable core may require several vertical scans to find a feasible location for the core. The first vertical scan is performed at the original horizontal location of the core's bitstream. For consecutive scans, the horizontal scan location is determined by adding the original horizontal location to the offsets contained in the offset group combatable with the core. Algorithm 6.2 shows the process of the full resource matrix scan. Once a location for a relocatable core that is scheduled for configuration is found, the resource matrix can be updated so that the matrix elements corresponding to this location are filled with logic '1' (used resources). Algorithm 6.3 can be used to fill a given area in the resource matrix with the desired value.

```
Struct location    find_Y_offset  (int X, int task_height, int task_width, int matrix_height,
                                  int matrix_width){

int Y,clear_count,count,index;        //local variable
struct location temp;                 // local instance of the structure 'location'

Y=0;                                  // vertical scan location
clear_count=0;                        // an internal flag to clear resource counter
temp.flag=0;                          // clear location flag
temp.X_offset=X;                      // initialize horizontal offset
while (Y<=matrix_height){             // start scanning the resource matrix vertically
        for (index=0;index<task_width;index++){   //scan the task width
                if (matrix[y][(index+X)]==1){
                        clear_count=1;            // set flag if used resource is found
                        break;
                }
        }
        if (clear_count ==1){                     // clear resource counter
                clear_count=0;
                count=0;
                Y=Y+1;                            // set vertical scan location
        }else                         //increment  scan location and resource count
                count= count+1;
                Y=Y+1;
                if (count==task_height){          // location found
                        temp.flag=1;
                        temp.Y_offset=(Y-task_height);
                        break;}
        } return temp; }                          //return location parameters
```

**Algorithm 6.1 Vertical scan of the resource matrix**

```
struct location find_location (int task_height, int task_width){

int index;                    //local variable
struct location temp;         // local instance of the structure 'location'

//scan vertically at the original horizontal location of the core
temp=find_Y_offset( original_horizontal_location, task_height, task_width);

if (temp.found==0){           //if no location found, start scanning at offsets in the offset group
        for (index=0;index<number_of_offsets;index++){
                temp=find_y_offset( (original_horizontal_location+offset_group[index]),
                                        task_h, task_w);
                if (temp.found==1){
                        temp.x_offset= original_location + offset_group[index];
                        break; }
                }
        }else { temp.x_offset= original_location;
}return temp;}
```

**Algorithm 6.2 Resource matrix scan**

```
void update_matrix (int X, int Y, int height, int width, int value){

int index1,index2;                    //local variables

 for (index1=0;index1<height ;index1++){              //vertical index
        for (index2=0;index2<task_w;index2++){    //horizontal index
                matrix [Y+index1][X+index2]=value;          //fill area with desired value
                                        }
}}
```

**Algorithm 6.3 Update resource matrix**

## Enhancing System Efficiency via Task Reuse

When location parameters are found for a particular task's core, the core needs to be configured on the FPGA before the task can start execution. In a basic placement scheme, the location occupied by a core assigned to a task that finished execution is updated as 'available' in the resource matrix so that the cores of future tasks can be placed in the same location. This placement scheme can lead to inefficient utilisation of the ICAP port when several tasks using the same relocate core are scheduled for execution.

It would be convenient if the placement algorithm keeps track of the already configured cores so that future tasks that use the same cores can be assigned directly. This circumvents the need for reconfiguring the cores and can result in a much more efficient system. To allow for task reuse, several modifications in the basic placement scheme are required.

First of all, a table containing information about the already configured cores that have finished execution is added. This table lists the type and number of the available cores along with their location parameters. The FPGA resource representation in the resource matrix is also modified. Three states are used to represent the state of the FPGA resources in the matrix, '0' for available resources, '1' for resources occupied by an active core and '2' for resources occupied by a free core.

Figure 6.6 shows the proposed placement scheme, which takes into account the already configured and available cores. When a task is scheduled for execution, the task allocation process goes through the following stages:

1) Scan the available cores table: the core required for the scheduled task is compared with the cores in the table, if one or several instances of the core are already configured on the FPGA the task is assigned to one instance. This instance is then removed from the core table and its location is filled with '1' in the resource matrix.
2) Scan the empty resources: if no core to fit the task is found in the previous stage, the resource matrix is scanned. Only resources marked with '0' in the resource matrix are considered available in this stage. If a feasible location is found for the core, the location is filled with '1' in the resource matrix.
3) Expand the search space: if no feasible location is found in the previous stage, the resource matrix is scanned again; however this time resources marked as either '0' or '2' are considered available. If a feasible location is found for the core, the location is filled with '1' in the resource matrix. If no feasible location is found at this stage, this means that the task cannot be configured on the FPGA until other active cores finish execution.

4) Check for location overlap: if a location is found in stage 3, this stage is triggered to check if the found location overlaps with one or more of the free cores. In case of a location overlap, the cores located at the overlap region are removed from the core table and any resource of these cores marked with '2' in the resource matrix are filled with '0'.

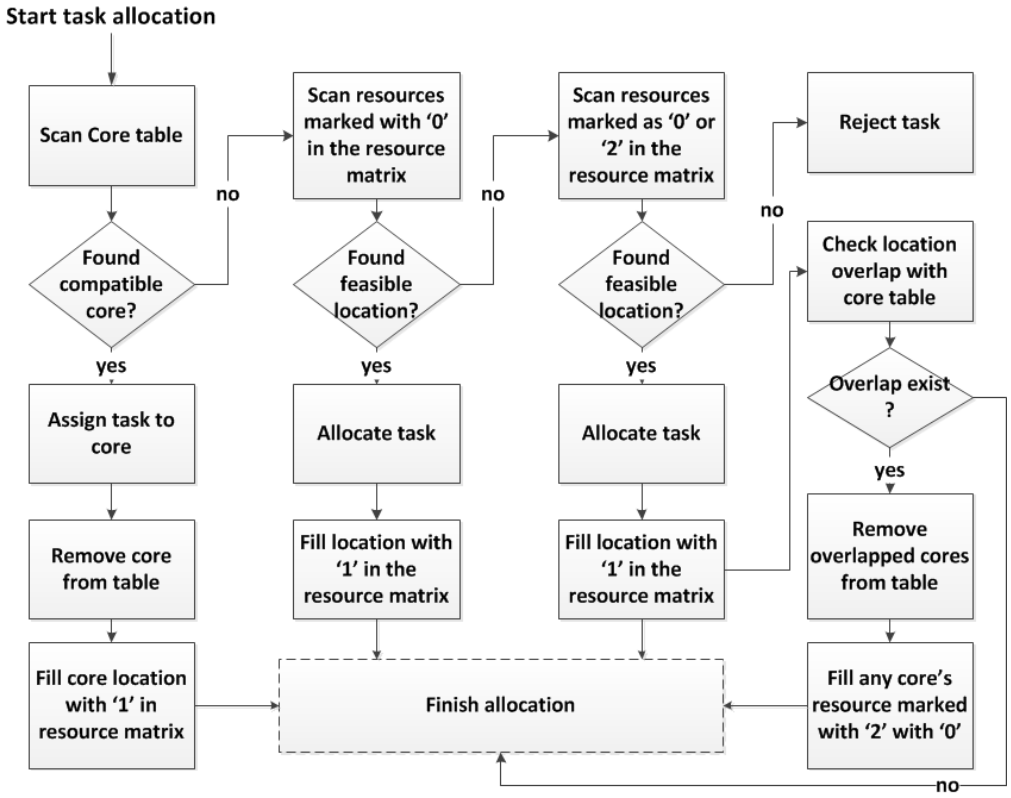



**Figure 6.6 Placement scheme with task reuse support**

## 6.2.3 Configuration Management and Task Execution

Secure configuration is an important requirement in any FPGA system, especially if the system is deploying cryptographic cores that are configured using externally stored bitstreams. As mentioned earlier, FPGAs offer bitstream encryption capabilities. A decryption block is usually integrated in the FPGA to decrypt the bitstream before configuration. This decryption block cannot be accessed by user-logic as it is coupled with the configuration logic of the device. The proposed system

deploys bitstream relocation to enable the configuration of the same bitstream across several locations on the chip. Bitstream relocation requires online modifications of the bitstream content (see Chapter 4). This means that an encrypted partial bitstream must be decrypted using a decryption block implemented on the FPGA logic before performing any modifications. The full configuration of the device, which contains the static components in the system, is loaded into the FPGA memory after power-up. The encrypted full bitstream can use the FPGA embedded decryption block. However, any configuration operation using the ICAP requires a different decryption block implemented in the system's static logic. Figure 6.7 shows the configuration process of encrypted relocatable partial bitstreams.

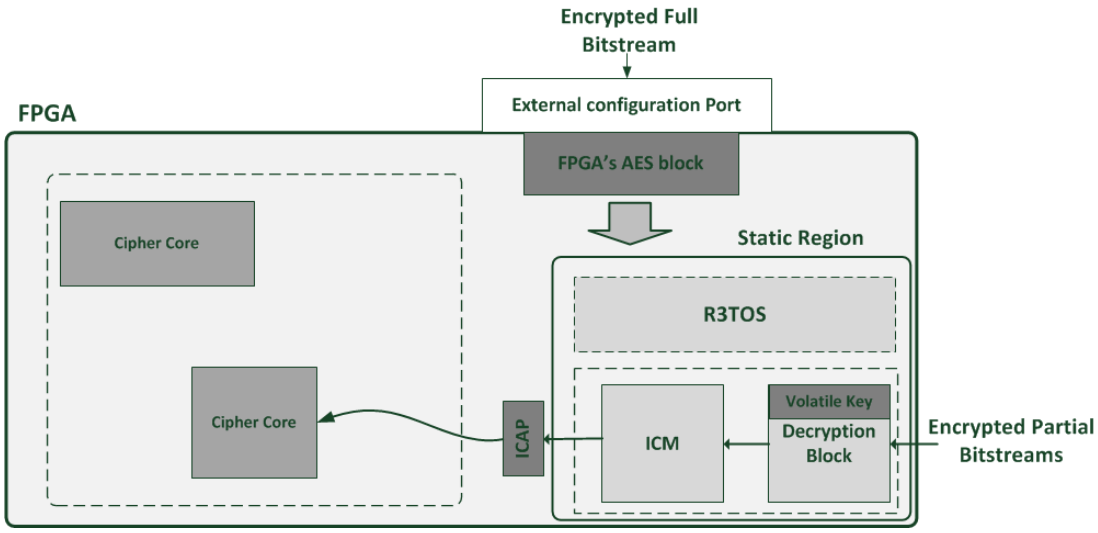



**Figure 6.7 Secure configuration of relocatable cipher cores**

In the proposed system, cryptographic tasks are organised in a queue according to their priority. Four states can describe the status of the tasks during the operation of the system: Waiting, Executing, Finished and Failed. The system uses the ICAP to perform different operations: task configuration, task data transfer, readback scrubbing and BIST diagnosis. Access to the configuration port must be managed carefully to allow efficient utilisation of the ICAP maximum bandwidth. Figure 6.8 describes the operation of the system and how access to the ICAP is managed between the different operations.

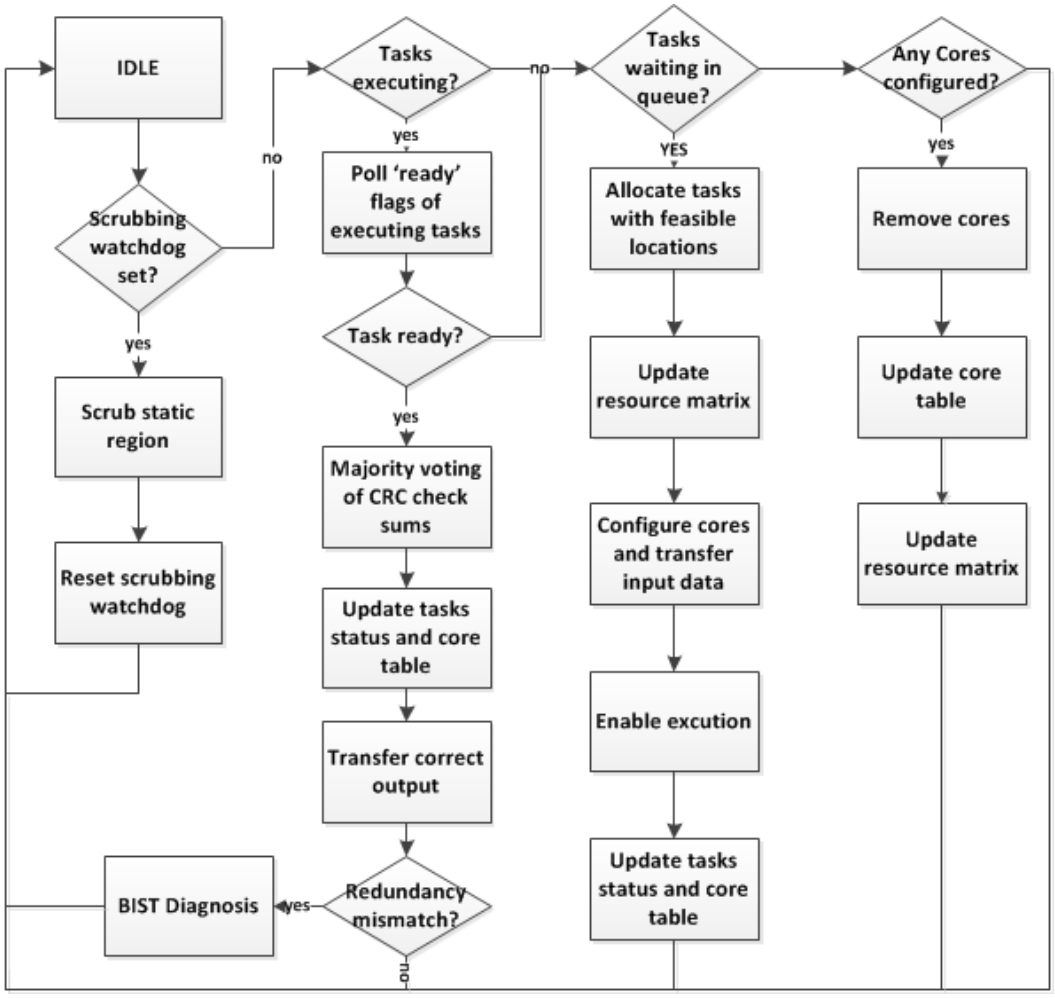



**Figure 6.8 Simplified operation of the system**

When the system is idle, different flags can trigger the system to exit the idle state and perform a specific operation. These operations are: 1) readback scrubbing of the static logic; 2) collecting the outputs of the tasks that have finished execution tasks; and 3) the configuration of the tasks' cores waiting in the tasks queue.

Readback scrubbing has the highest priority of the three operations. Readback scrubbing is the primary method of fault recovery in the static logic of the system. A watchdog timer is used to generate a scrubbing request, which triggers the scrubbing operation from the idle state. When the scrubbing operation is complete, the watchdog timer is reset and the system returns to the idle state.

If a scrubbing request is not set and a flag indicates that there are tasks already configured in the system, readback operations through the ICAP are performed to check the status of the configured tasks according to their order in the task queue. If any of the configured tasks has finished execution, the system starts the process of collecting the task output from its output buffer. As there are three redundant modules of each task, majority voting is performed to determine if one of these modules has failed during its execution. To accelerate the process of voting, only the checksum buffers of each redundant module are read through the ICAP. This requires a single configuration frame to be read from each module rather than reading the entire output buffers of the modules. If no error is determined by the voting process, the output is collected from one of the redundant modules and the status of the task is updated. On the other hand, if a voting process has showed that one of the three modules is faulty, the output is collected from one of the intact modules and a diagnosis operation is performed to determine if the cause of the fault is a damaged FPGA resource (see Chapter 5). If a damaged resource is found in the diagnosis process, this resource is marked as 'used' in the FPGA resource matrix so that it is circumvented when allocating new tasks for execution.

The third main operation performed by the system is the configuration of tasks waiting in the task queue. This operation starts by performing the placement algorithm to find three feasible locations for each task in the queue. Allocated tasks are then configured and their input load is transferred. The FPGA resource matrix is also updated to fill the locations occupied by the tasks. Because three instances of each task are configured, this stage results in the heaviest load on the ICAP port. It is important to try to reduce the configuration speed as much as possible in this stage. The multiple-clone configuration technique can dramatically reduce the configuration time if the same core is to be configured on several locations on the FPGA. Figure 6.9 shows the multiple-clone configuration scheme used when several tasks using the same core are scheduled for configuration.

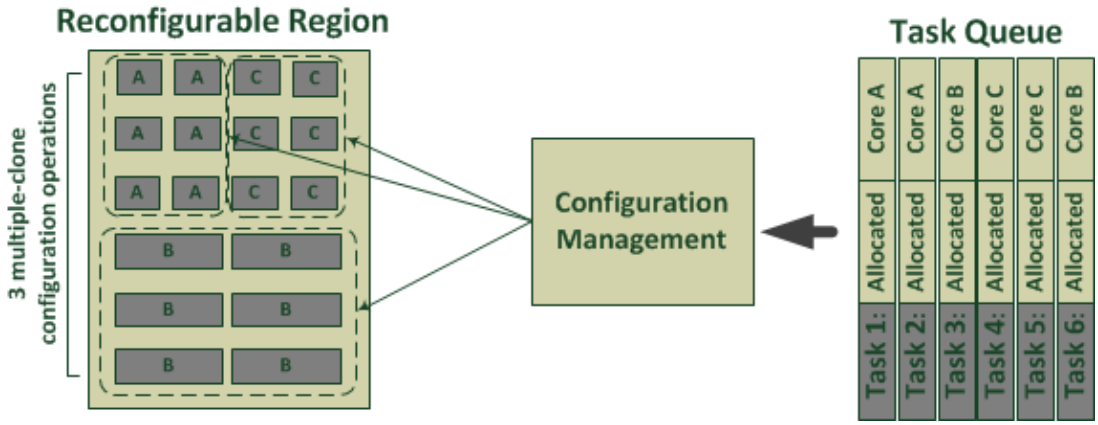



**Figure 6.9 Multiple-clone configuration of the same cipher core**

## 6.3 Proof-of-Concept Implementation

A prototype of the proposed system was implemented on a Virtex-4 FX60 FPGA. The implementation consists of two parts: the first part is the design of a test relocatable cipher. The second part is the static control system which is a reduced version of the R3TOS.

### 6.3.1 Implementation of a Test Relocatable Cryptographic Core

To test the scalability of the proposed system, the 'PRESENT' block cipher was modified as a relocatable core. The PRESENT cipher is a relatively new light weight block cipher especially designed for systems requiring low power consumption and low resource utilisation [174]. The PRESENT cipher was developed at the University of Leuven (Netherlands) in calibration with Orange Labs (France), Ruhr University Bochum (Germany) and Technical University of Denmark. Recently, it has been included as a light weight cryptography standard by the International Organization for Standardization and the International Electro-technical Commission. An open-source optimised VHDL implementation of the PRESENT cipher is used for the cipher block in the relocatable core (available in [175]). The

PRESENT cipher operates on 64-bit blocks and uses an 80-bit key. The cipher requires 32 clock cycles to finish encrypting a single block of plain text. Figure 6.10 shows a high-level block diagram of the PRESENT cipher. The resource utilisation of the cipher when implemented in a Virtex-4 FPGA is shown in Table 6.1.


**Figure 6.10 The PRESENT cipher block diagram [174]**

**Table 6.1 Resource utilisation of the PRESENT cipher in a Virtex-4 FPGA**

| Resource Type | Utilisation |
|---|---|
| ***Slices*** | 158 |
| ***LUTs*** | 236 |
| ***BRAMs*** | 0 |

A relocatable core based on the PRESENT cipher was designed for implementation in a Virtex-4 FPGA following the architecture shown in Figure 6.2. To modify the cipher as a relocatable core, the input/output BRAM blocks and LUTs are added and constrained to specific locations within the core. The size of plain text that can be transferred into the core will depend on the size selected for the input/output BRAM blocks inside the core. This means that larger text should be divided into several segments. These segments can be encrypted sequentially using the same core or concurrently using several cores initialised with the same key. On the other hand, plain text smaller than the BRAMs size in the core requires data padding as the core operates on a fixed size of block.

In the implemented design, a single BRAM column containing four blocks is used for the data transfer. The size of each BRAM block is 2KB. Two of these BRAM blocks are used to store the key and the plain text and the other two are used to store

the cipher text. Figure 6.11 shows the data mapping in the cipher core's BRAM blocks.

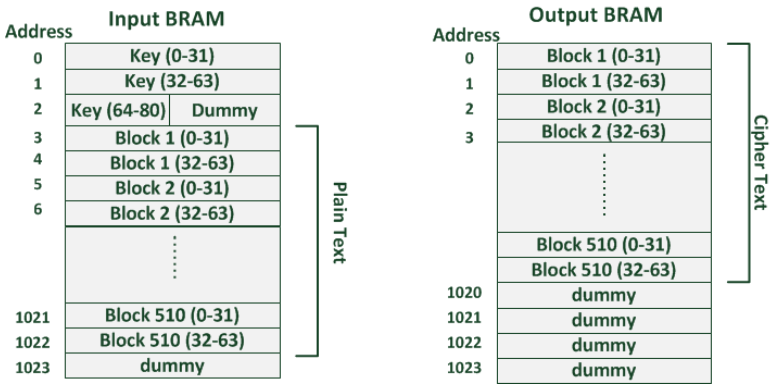



**Figure 6.11 Data mapping in the cryptographic core**

An LUT is used to pass the core enable signal from the R3TOS kernel to the core using the ICAP, and another LUT is used to store a 'ready' flag indicating that the core has finished execution (see Figure 6.12).

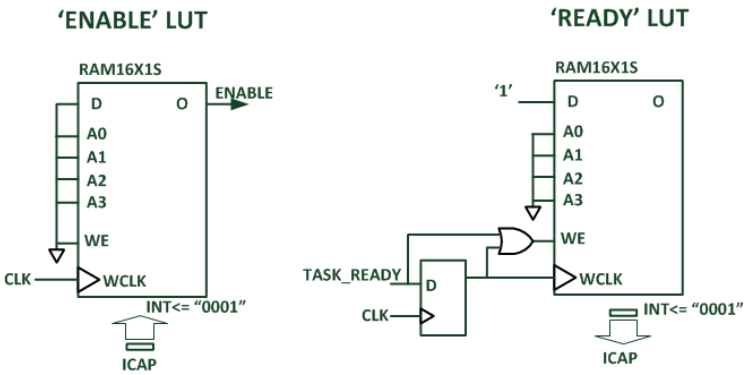



**Figure 6.12 Relocatable core's LUT semaphores**

As the size of data handled by the core is relatively small, a parallel CRC-16 generator is added into the core to generate the CRC checksum for the cipher text. It

is possible to store the output of the CRC generator in a single LUT. This requires dedicated control logic to write each bit individually to the LUT (see Figure 16.13).

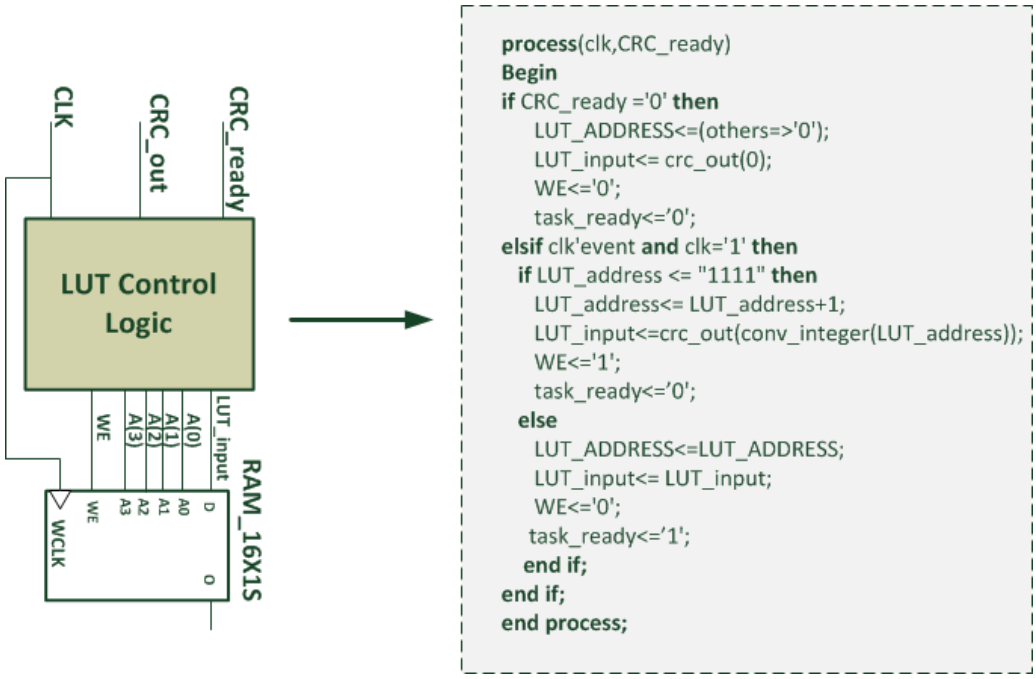



**Figure 6.13 Checksum LUT**

The implementation of a relocatable core based on the PRESENT cipher has resulted in an approximate 53% increase in the slice utilisation compared to the original cipher. Table 6.2 shows the resource utilisation of the relocatable cipher in a Virtex-4 FX60 FPGA.

**Table 6.2 Resource utilisation of the relocatable cryptographic core in a Virtex-4 FPGA**

| Resource Type | Utilisation |
|---|---|
| ***Slices*** | 242 |
| ***LUTs*** | 417 |
| ***BRAMs*** | 4 |

Only a single partial bitstream is generated for testing the core in the proposed system. The core layout is shown in Figure 6.14. It can be seen that the core occupies six CLB columns and a single BRAM column. The layout selected for the relocatable cipher is compatible with 'offset group B' in Figure 6.4.

The ENABLE and READY LUTs are all placed on the bottom of the first CLB column in the core (see Figure 6.14). To be able to modify the content of these LUTs using the ICAP, knowledge of the configuration bits that correspond to the values stored in these LUTs is required. Reverse-engineering experiments have revealed the exact locations of these configuration bits in a Virtex-4 FPGA. In any Virtex-4 FPGA, the 19$^{th}$ and 21$^{st}$ frames contain the content of the column's LUTs. More precisely, the 19$^{th}$ frame contains the content of SLICE-M LUTs and the 21$^{st}$ frame contains the content of SLICE-L LUTs. Since only SLICE-M can be used as distributed RAM, the relocatable core's LUTs are all placed on SLICE-M LUTs of the first CLB column. Figure 6.15 shows how these LUTs are mapped into the configuration bits of the 19$^{th}$ frame of the column.

It is also noted that the relocatable cipher connects to a regional clock buffer by default. Since the height of the cipher core is equal to a single clock region (one column), any regional clock buffer can be enabled and disabled using the ICAP to freeze the operation of cores placed in the clock region. Disabling the core's clock prior to accessing its LUT using the ICAP is important to prevent corrupting its content (see Chapter 5).

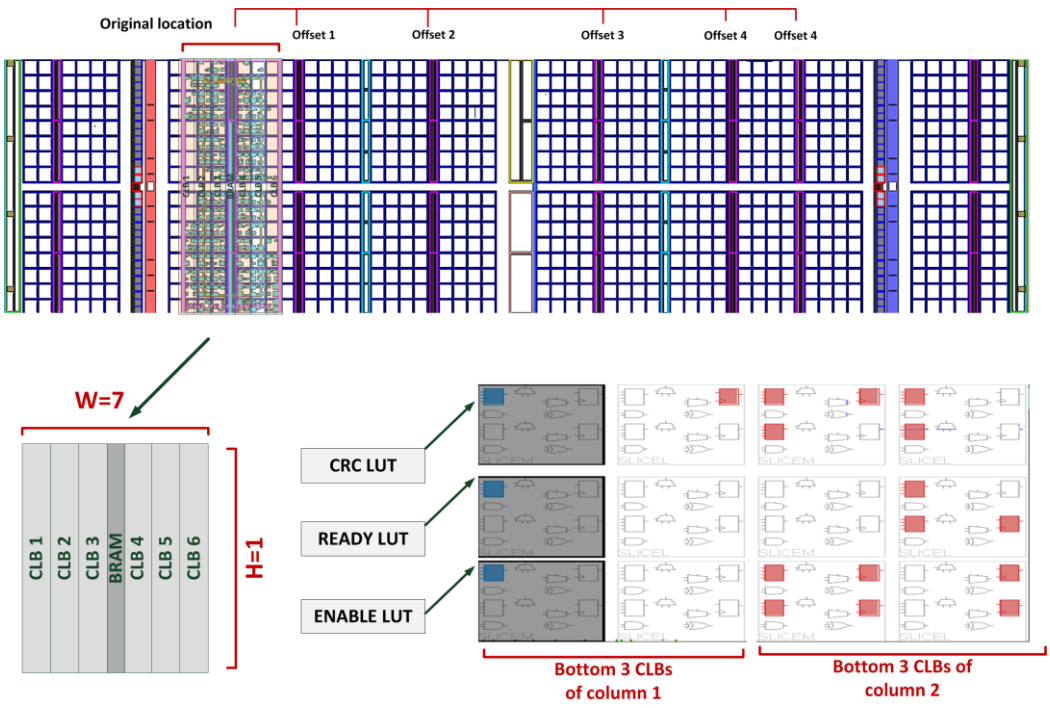



**Figure 6.14 Resource layout of the relocatable cipher**

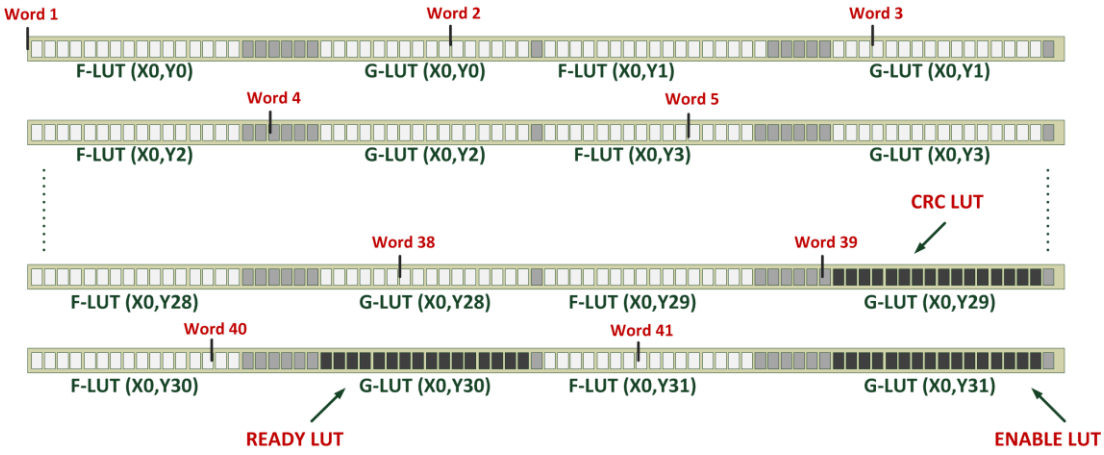



**Figure 6.15 15 Input, output and CRC LUTs mapping in the 19th frame of the first CLB column**

## 6.3.2 Implementing the Static Control Logic

The static control logic in the system performs the functionality described in Figure 6.8. Figure 6.16 shows the main components of the control logic in the implemented system.

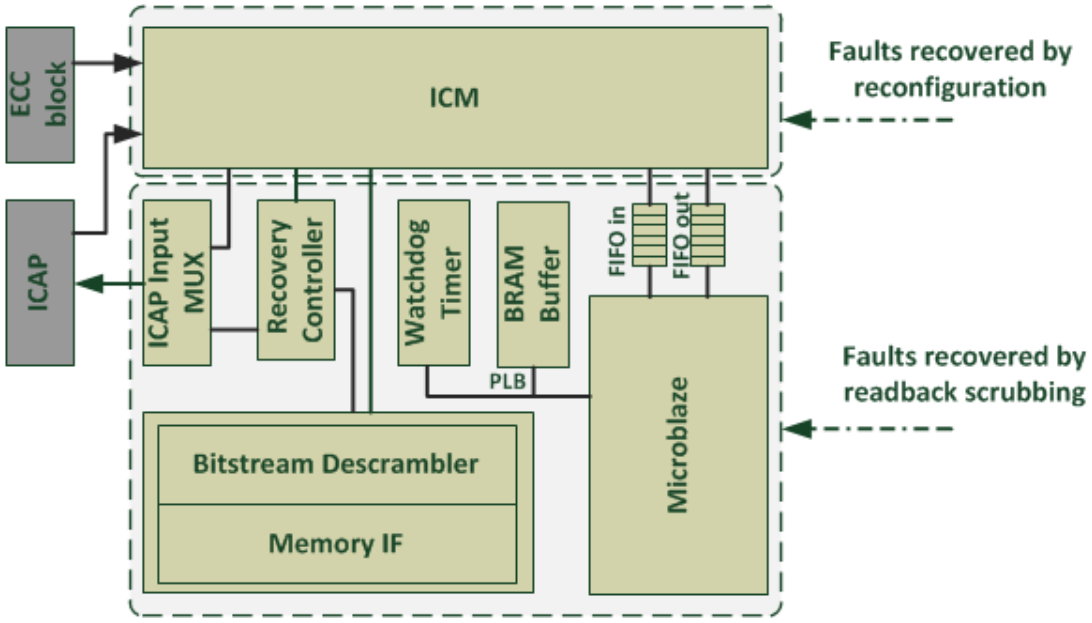



**Figure 6.16 Simplified diagram of system's components**

A Microblaze processor is used to run the software of the system and the smallest version of the FT ICM is used to manage the configuration of the relocatable cipher cores (see Chapter 5). The Microblaze communicates with the ICM using two FSL connections, which are basically two FIFOs; one is used to send configuration instructions and the other is used to monitor the status of the configuration operations.

The ICM logic is split between two regions. The first is a reconfigurable region, which contains the full ICM. The full ICM performs the main configuration operations such as: core configuration, core data transfer and readback scrubbing. The second region contains a recovery control alongside the static components in the system. The ICAP input connections are multiplexed between the ICM and the recovery controller, which perform DPR of the ICM to recover any faults causing configuration errors. When the ICM is active, readback scrubbing cycles are performed at particular rates to recover faults in the static region of the system. The scrubbing cycle's rate is determined by a watchdog timer, which is connected to the Microblaze processor via the PLB bus.

The Microblaze processor is also connected to a BRAM buffer, which is separate from the main memory. This buffer is dedicated for the transfer of the core's input and output data. The core's input data transfer is accomplished by reading the configuration frames of the BRAM buffer, which contains the input data via the ICAP, and then writing these frames to the core's input BRAM. This operation is reversed for the core's output transfer. Because the content of each BRAM block maps into 64 configuration frames, large ICAP read/write operations are required to reduce the transfer time overhead. To allow for large ICAP read and write operations, the data buffer size in the ICM was increased to six BRAM blocks.

As mentioned earlier in this thesis, partial bitstreams stored in external memory should be encrypted. Ideally, AES should be used for partial bitstream encryption; however, in this proof-of-concept implementation, only a small parallel data descrambler is used, which has a very small area footprint and results in only one clock cycle delay in the configuration (available in [176]). The data descrambler is integrated with the ZBT-SRAM memory controller to automatically decrypt any data fetched from external memory according to the polynomial: $X^{16} + X^5 + X^4 + X^3 + 1$.

Figure 6.17 shows a floor-plan image of the implemented control logic. The control logic is constrained to the top-left corner of the chip and spans four clock regions vertically. The RP containing the ICM is placed in the first clock region and the rest of the control logic is placed in the other three clock regions. All the regional clock buffers are instantiated and enabled with a default frequency of 100MHz.

All the local routes of the control logic are confined within the top-left region of the chip. The area occupied by the control logic accounts for around 25% of the chip, leaving most of the chip free and available for the placement of the relocatable cipher cores apart from a few columns that contain the static routing to some of the system's IOs.

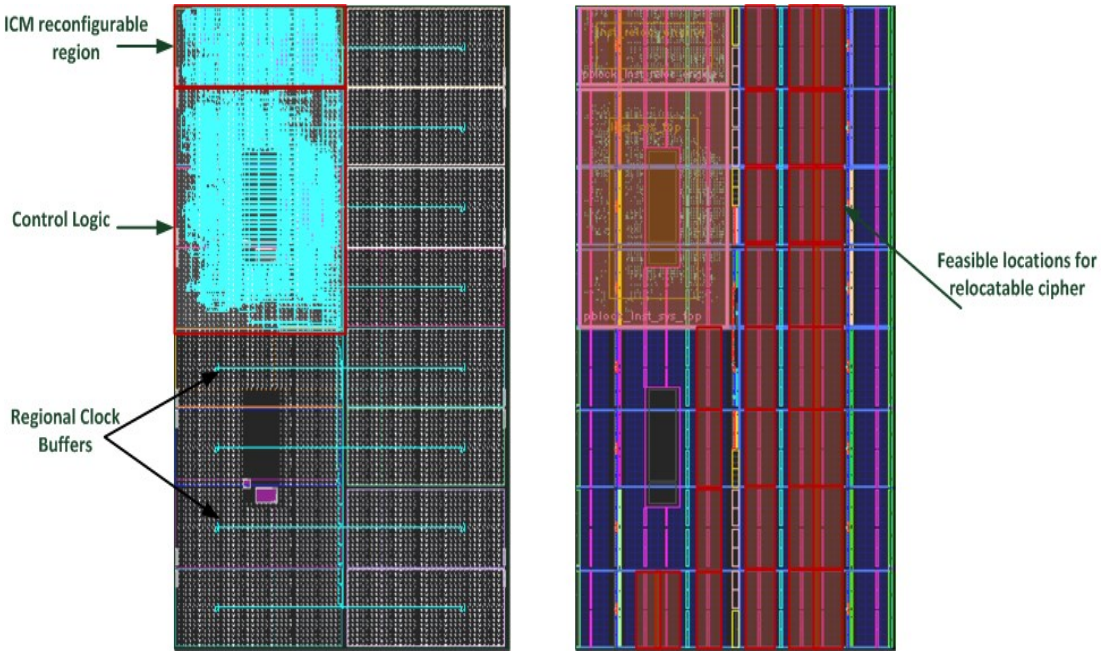



**Figure 6.17 Floor-plan image of the control logic in the system**

With the implementation shown in Figure 6.17, a total number of 30 feasible locations are possible for the relocatable test cipher. However, only 21 cores can be configured on the chip at one time. Figure 6.18 shows the initialisation of the resource matrix used in the placement algorithm. The matrix consists of 65 columns and eight rows. The matrix is initialised so that unavailable resources are mapped with logic '1' in the matrix. These resources are the resources occupied by the control logic, resources with static routes passing through them, hardwired components on the chip and resources with types not used by any core in the system. In the test application, both the DSP columns and the clock resources columns are mapped with '1' in the resource matrix as they are not used by any core in the system. In addition, the locations occupied by a PowerPC processor that is integrated in the Virtex-4 FPGA are also mapped with '1' in the resource matrix.

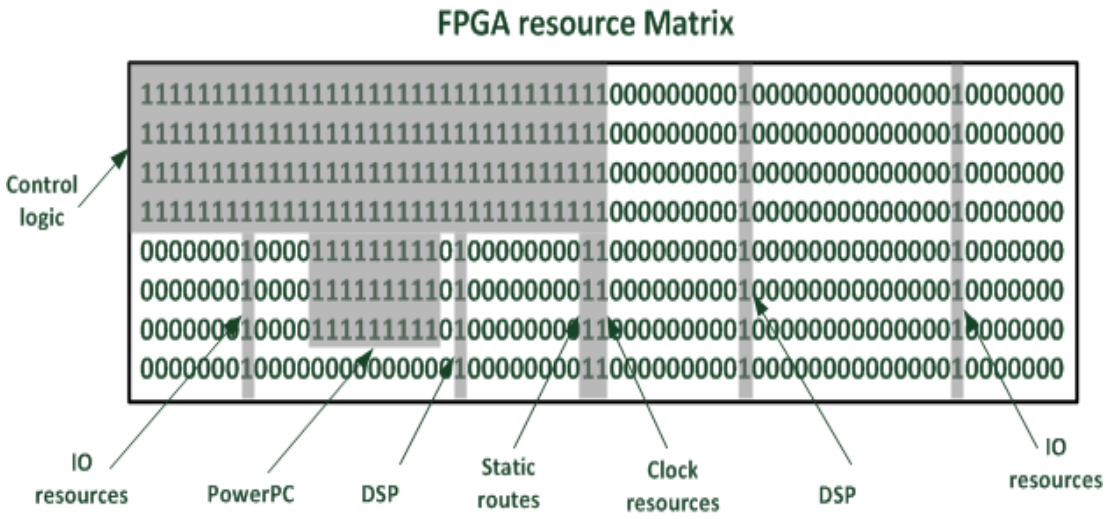



**Figure 6.18 Initialisation of the FPGA's resource matrix**

The placement algorithm described earlier generates two offsets when a feasible location is found for a particular core: the horizontal X_offset and the vertical Y_offset. After applying the placement algorithm on the matrix shown in Figure 6.18, the generated Y_offset is compatible with the ICM's relocation offset format and can be used directly for relocation (see Chapter 4). This is not the case for the horizontal offset as it is not compatible with the FPGA's column addressing and the ICM's relocation offset format, which requires two different offsets, one for the CLB resources and the other for the BRAM resources. This horizontal offset can be easily converted to the required format depending on the selected offset in the offset group:

Original location → CLB_offset= 0 , BRAM_offset= 0

Offset (index) → CLB_offset= Offset (index) – index , BRAM_offset= index

## 6.4 Experimental Results

This part of the thesis reports the performance of the implemented system when executing tasks based on the test cipher core. The system clock frequency was set to 100 MHz and the scrubbing rate was set to one cycle every 10ms. Although any scheduling algorithm is feasible with the proposed system, a simple First-Come-

First-Served (FCFS) scheme was assumed so that the first incoming task is scheduled first for execution. The experimental analysis assumes that tasks along with their input load are available in the CPU's main memory. No external IO overhead is considered for encryption quarries.

### 6.4.1 Task Allocation

When an encryption task is scheduled for execution, three redundant modules of the cipher core are configured in three different locations on the reconfigurable area on the FPGA. This allocation scheme requires the placement algorithm described earlier to be executed three times. Any task that does not fit in a minimum of three feasible locations is not allocated and has to wait for other active tasks to finish execution.

The time overhead for task allocation will depend on the status of the tasks in the reconfigurable area. The placement algorithm goes through three stages. The first is scanning the 'core' table that contains the status of already configured cores in the FPGA. Tasks with three feasible locations in this stage are allocated very quickly as the location offsets required for the tasks' cores are already calculated. The second stage scans the available resources in the resource matrix to find a feasible location for the task's core. So, three scans of the resource matrix are required for the second stage, one for each of the task's redundant cores. The third stage of the placement algorithm scans the empty resources as well as resources occupied by tasks that have finished execution. Since there is only one core type in this test application, the placement algorithm will never go through the third stage. Table 6.3 shows the time overhead breakdown for task placement in the system.

**Table 6.3 Task allocation time overhead**

| Operation | Min. Time (us) | Max. Time (us) |
|---|---|---|
| ***Core Allocation*** | 1 | 42 |
| ***Task Allocation (3 cores)*** | 5 | 125 |

## 6.4.2 Configuration and Control of the Relocatable Cores

The configuration scheme used in this test application deploys the multiple-clone configuration technique as described in Figure 6.9. In Virtex-4 FPGAs, a single cloning operation can only cover cores allocated to the same half of the FPGA chip. The vertical offsets for the allocated tasks are scanned to determine the number of operations required to finish the configuration process. A maximum of two configuration operations is required, one for each half of the FPGA. Figure 6.19a shows the configuration time overheads of the test cipher cores when varying the number of allocated tasks in the placement stage, and Figure 6.19b shows the number of cores configured in each configuration operation.

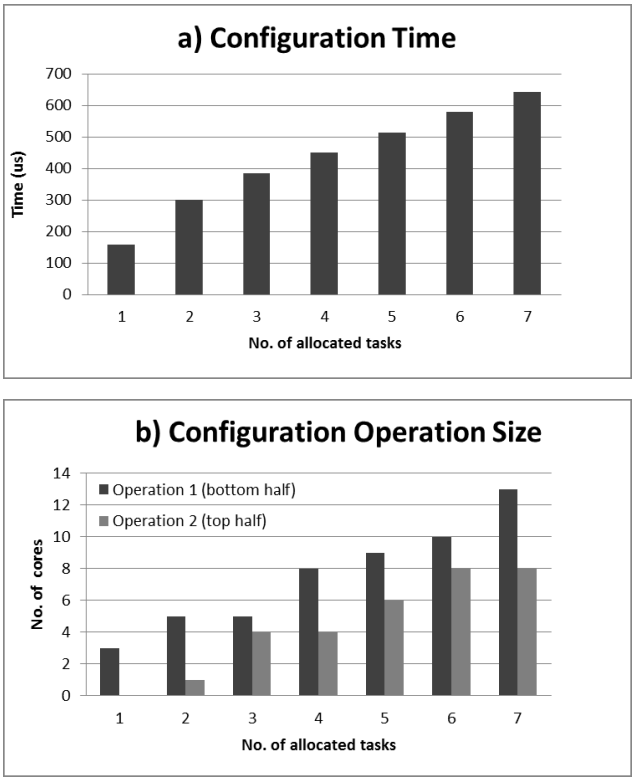



**Figure 6.19 Task configuration**

When cores need to be removed from the reconfigurable area, a black-box bitstream is configured on top of them. This bitstream contains only empty frames and, hence, only a single cloning operation is required for the removal of any number of cores in the FPGA. Figure 6.20 shows the removal time overhead for different numbers of tasks.

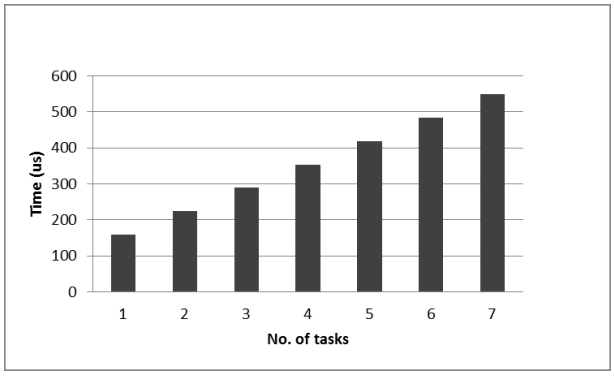



**Figure 6.20 Task removal time**

To control and monitor the operation of a core after configuration, several operations are performed using the ICAP: enable/disable the regional clock buffer, enable/reset the core and monitor the status of the core. For the first two operations, a field in a certain configuration frame needs to be modified (LUT content). To do this, an ICAP read operation is performed followed by an ICAP write operation. To monitor the status of an active core, only a read operation is required to check the value of the READY LUT. Table 6.4 shows a time overhead break down of the operations required to perform the control operations. It is noted that any of these operations are repeated three times for each task.

**Table 6.4 Task control time overhead breakdown**

| Operation | Time (us) |
|---|---|
| *Readback frame via ICAP* | *7.5* |
| *Fetch word containing the required field to Microblaze* | *5.8* |
| ***Total time to check a field in a frame*** | ***13.3*** |
| *Send modified word to ICM* | *5.8* |
| *Write frame containing field via ICAP* | *7.5* |
| ***Total time to modify a field in a frame*** | ***26.6*** |

### 6.4.3 Task Data Transfer

To transfer the input load to an already configured core, the Microblaze first writes the input load to the dedicated BRAM buffer. The ICM performs a read operation of 64 frames to store the configuration of the BRAM buffer before performing another write operation of 64 frames to the core's input buffer. The ICAP write operation is repeated three times, one for each redundant core of the task. Table 6.5 shows a breakdown of the task input data transfer time overhead.

**Table 6.5 Task input data transfer time**

| Operation | Time (us) |
|---|---|
| *Transfer input load from Microblaze to BRAM buffer* | 92.7 |
| *Read the configuration of the input buffer via ICAP* | 33.3 |
| *Write the copied 64 frames to core's input BRAM via ICAP* | 33.3 |
| ***Total input data transfer time (3 cores)*** | **225.9** |

When a task's cores have finished execution, checksum voting is performed and the correct output is transferred to the Microblaze. First, the three frames containing the core's checksums are read and transferred to the Microblaze, which performs the voting. One of the correct outputs is then read via the ICAP and copied to the BRAM buffer. Table 6.6 shows a breakdown of the task output data time overhead.

**Table 6.6 Task output data transfer time**

| Operation | Time (us) |
|---|---|
| *Read frame containing CRC-checksum* | 7.5 |
| *Fetch word containing CRC-checksum* | 5.8 |
| ***Total voting time*** | **40** |
| *Read correct output via ICAP* | 33.3 |
| *Transfer output from BRAM buffer to MB* | 92.7 |
| ***Total output transfer time*** | **166** |

### 6.4.4 Fault Detection and Recovery

Faults affecting the static control logic are recovered by readback scrubbing, whereas faults affecting the ICAP controller are recovered by DPR (see Chapter 5). When a scrubbing cycle is due, readback scrubbing is only performed on the region occupied by the control logic, which spans a total of 1,716 configuration frames. The scrubbing cycle is performed using three readback operations one for each row in the region occupied by the control logic. On the other hand, the region occupied by the ICM is reconfigurable with a partial bitstream of a size equal to 155 KB. The time overhead for detecting faults in a region occupied by a faulty core is dictated by the BIST diagnosis process of the CLB columns. Table 6.7 shows the fault recovery and detection operations time overheads.

**Table 6.7 Fault detection and recovery time overhead**

| Operation | Time (us) |
|---|---|
| *Readback scrubbing of control logic* | 805 |
| *ICM reconfiguration* | 404 |
| *BIST Diagnosis* | 1231.9 |

### 6.4.5 Task Execution Time Overhead

In this analysis, the execution time of a relocatable core is defined as the time required for the core's output to be available in the main memory after it finishes execution. Table 6.8 shows the execution time of the test cipher core with and without the redundancy and compares it to a software implementation of the test cipher in a Microblaze processor where temporal redundancy is used.

**Table 6.8 Test cipher core execution time**

| Execution scheme | Time (ms) | | Speedup |
|---|---|---|---|
| | Proposed system | Software | |
| ***With TMR*** | 1.3 | 2912.7 | x2240.5 |
| ***Without TMR*** | 0.5 | 970.9 | x1942.8 |

Table 6.8 shows the task when there is only one task in the queue and no cores are already configured in the reconfigurable region. In the proposed system, the number of tasks in the queue and the status of the reconfigurable region will affect the time required for a task's output to be available in the main memory. In a FCFS scheduling scheme, the wait time is defined as the time a task remains in the queue waiting for execution. In FCFS scheduling, the last task in the queue will always have the longest wait time. Figure 6.21 shows the maximum task wait time in the proposed system for the different number of tasks in the queue, all with the same arrival time.

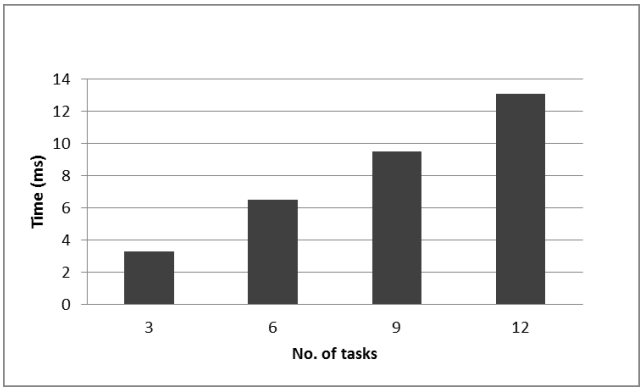



**Figure 6.21 Maximum task wait time**

## 6.5 Chapter Conclusion

Reliability is very much related to the security of cryptographic hardware as faults can compromise the security of the system. Traditional FT design techniques are usually deployed in cryptographic systems to prevent the leaking of secret information as a result of faults. These techniques, however, limit the flexibility and prevent the efficient deployment of the FPGA's resources. The R3TOS provides a flexible platform for implementing reconfigurable applications that require flexibility and high performance. This makes R3TOS an ideal solution for implementing reliable cryptographic applications. This chapter presented the design and

architecture of an R3TOS-based encryption engine. This encryption engine is capable of executing real- time encryption tasks using relocatable cipher cores. These cores are efficiently allocated in the available resources of the FPGA using a new placement scheme that accounts for the irregularity in the FPGA fabric and allows for the placement of heterogynous cores. This placement scheme also allows for intelligently reusing the already configured cores to enhance the efficiency of the system. Faults in the static logic of the system are mitigated by means of readback scrubbing, whereas cipher tasks are executed with modular redundancy to ensure secure operation. A proof-of-concept implementation of the system in a Virtex-4 FPFA was demonstrated and tested with a test relocatable cipher core. The test results showed that HTs based on the test cipher outperform software tasks of the same cipher.

# Chapter 7 : A DPR-based Platform for Frequent Itemset Mining Acceleration

Knowledge Discovery and Data Mining (KDD) is a growing field focusing on extracting useful information from large amount of data. KDD is applied in several fields such as science, medicine and business. One important concept in data mining is Frequent Itemset Mining (FIM), which is widely deployed for extracting information from businesses and enterprise databases. FIM is often used in market basket analysis to understand the purchase behaviour of customers purchasing products offered by the same retailer. Usually, customer purchase information is stored in a transaction database which consists of several transactions; each transaction contains the products purchased by a single customer. Association rules can be derived from the databases to see how often certain products are bought together. These association rules can influence business decisions to increase future profit. Retailers can use the information extracted form mining their databases to come up with the best pricing, promotional offers and store layout. FIM is not limited to the basket analysis. FIM concepts can be used in other applications such as web mining and bioinformatics.

Using FPGAs, FIM can be accelerated by performing some acceleration tasks in hardware. Placing static accelerators in the system will reduce the resource efficiency and limit the size of each accelerator. DPR can be used to allow for sharing the FPGA's resources among the different acceleration tasks and consequently enhance the system's performance. This chapter proposes a novel system to accelerate the popular FP-growth algorithm using FPGAs. The system manages the configuration and execution of several acceleration tasks that utilises relocatable systolic array

accelerators. This chapter presents the design and architecture of the system and evaluates the performance gain archived by using DPR.

# 7.1 Background on Frequent Itemset Mining

Suppose we have a set of items $B=\{i_1,i_2,\ldots i_m\}$

625. Any subset of items collected from B is called an itemset. In the context of basket analysis, an itemset is a group of items that can be bought together.

A transaction over B is a set t= (ID, J) where ID is the unique transaction identifier and $J \epsilon B$. In basket analysis, J can be a list of products purchased by a single customer.

A transaction database is a collection of transactions T={t1, t2, …..,tn}. Every transaction is an itemset, but some items may not appear in T. A transaction can be decomposed into smaller itemsets, for example itemset I is covered by t=(ID, J) if $I \epsilon J$. The number of itemsets covered by a transaction is equal to $2^n$ -1 where n is the number of items within the transaction. For example, if a transaction t=ABC, the number of itemsets covered by the transaction is $2^3$ -1 = 7; these itemsets are A, B, C, AB, AC, BC and ABC.

FIM is based on fining how common itemsets appear in the database. A count number for each itemset in the database is calculated. The count number is called the 'support'. The support of a particular itemset indicates the number of transactions containing this itemset. The support is usually given as a percentage of the total number of transactions.

A 'support threshold' is specified at the beginning of the mining process; the support threshold indicates the minimum support for an itemset to be considered for generating the association rules. Any itemset with a support count less than the specified threshold can be discarded to reduce the complexity of the mining process. Table 7.1 shows an example of a transaction database T ={t1, t2, t3 ,t4 ,t5 ,t6} over a

set of items B={A, B, C, D}. Table 7.2 shows all the possible itemsets derived from B along with their support count.

**Table 7.1 Example database**

| Transaction ID | Items |
|---|---|
| *1* | A, B |
| *2* | C |
| *3* | A,B,C |
| *4* | B,C |
| *5* | A,B,D |
| *6* | C,D |

**Table 7.2 Support count for itemsets in the example database**

| Itemset | Support | Itemset | Support | Itemset | Support |
|---|---|---|---|---|---|
| A | 3 (50%) | AC | 1 (16.66%) | ABC | 1 (16.66%) |
| B | 4 (66.66%) | AD | 1 (16.66%) | ABD | 1 (16.66%) |
| C | 4 (66.66%) | BC | 2 (33.33%) | ACD | 0 (0%) |
| D | 2 (33.33%) | BD | 1 (16.66%) | BCD | 0 (0%) |
| AB | 3 (50%) | CD | 1 (16.66%) | ABCD | 0 (0%) |

Suppose that the support threshold is 50%, which means that an itemset is required to appear in at least half of the transactions to be considered in the association rules. The itemsets which satisfy this condition are: A, B, C, and AB. These itemsets are referred to as ‘frequent’ itemsets.

### 7.1.1 Background on FIM Algorithms

The concept of FIM was first introduced by Agrawal in [177] wherein an algorithm for finding frequent itemsets was used to derive association rules for the market basket model. Later Agrawal improved the algorithm and called it the Apriori algorithm ([178] and [179]). The Apriori aims at reducing the mining time for large

databases by exploiting the anti-monotonicity property of itemsets: 'if an itemset is found to be infrequent, any superset of this itemset is infrequent'. This property is used in a mining process called 'candidate generation', which repeatedly generates larger candidate itemsets from smaller frequent itemsets. Only the support count of these candidates is considered when scanning the database. By repeatedly scanning the database and increasing the length of the candidates in each scan, the algorithm stops when no more candidates satisfy the minimum support count.

The Apriori algorithm suffers from two main drawbacks. The first is that multiple scans of the database are required to find the frequent itemsets. The second drawback is the delay caused by the candidate generation process. These drawbacks could lead to intolerable time overhead when mining large databases. The most outstanding improvement over the Apriori is the FP-growth algorithm, which only requires two database scans and eliminates the need for candidate generation [180]. The FP-growth is based on transposing the transaction database into a compressed tree structure called the FP-tree. The FP-tree is then mined in a shorter time compared to the original database.

**The FP-growth algorithm**

The FP-growth is based on representing the database with an FP-tree which contains several branches each with several nodes. Each path in the tree represents an itemset in the database whereby the nodes encountered in this path are the items in the itemset (see Figure 7.1). A node in the FP-tree can be a 'parent' node to several 'children' nodes. Each node contains an item identifier along with a count number representing the support of any itemset from a path ending with this node. The FP-tree of a database can be created by the following steps:

1) The database is scanned to calculate the support count of each item in the database. A list of frequent items is created with infrequent items removed to reduce the size of the tree.
2) Items within each database transaction are sorted in descending order of the support count, with infrequent items removed from each transaction.

3) A root for the tree is created and labelled with NULL. The first transaction in the database is represented with a branch from this root. Each item in the transaction is represented by a node, so that each node is placed in a different level away from the root. For consecutive transaction, the first item of the transaction is compared with nodes in the first level. If a common node is found, the support count of this node is incremented and the next item is compared with the children nodes in the next level. If no common node is found, a new node is created and the remaining items of the transaction form a new branch from this node. When the FP-tree is created, a header table is built containing links between nodes containing the same item.

Figure 7.1 illustrates the steps required to create the FP-tree for the database in Table 7.1, with a 25% minimum support threshold.

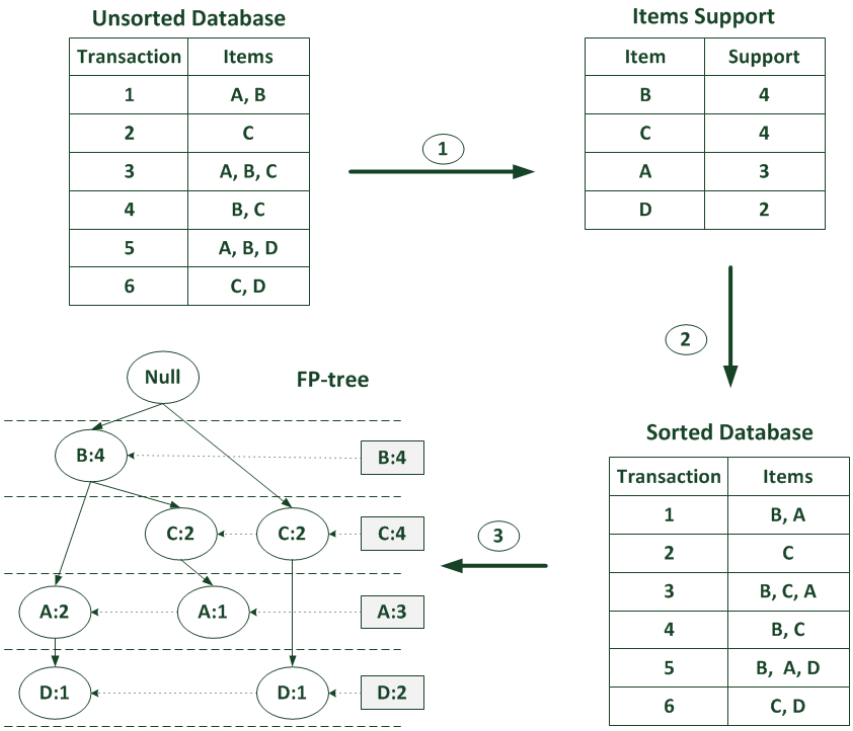



**Figure 7.1 Creating the FP-tree**

After creating the FP-tree, the mining problem is transformed into mining the tree rather than the whole database. By traversing the tree, a 'conditional pattern base' for each item can be created. The conditional pattern base represents the paths to the nodes containing the item. The itemsets containing the item can be found by the union of the item and the sub-itemsets in each path. The resulting itemsets are processed to add the support of similar itemsets. This procedure is repeated for all the items to generate all the possible itemsets. Table 7.3 shows the conditional pattern base and the possible itemsets for the FP-tree in Figure 7.1.

**Table 7.3 Itemsets generated from the FP-tree**

| Item | Conditional Pattern Base | Itemset |
|---|---|---|
| D:2 | {(BA:1),(C:1)} | BAD:1, BD:1, AD:1, CD:1, D:2 |
| A:3 | {(B:2), (BC:1)} | BA:3, BCA:1, CA:1, A:3 |
| C:4 | (B:2) | BC:2, C:4 |
| B:4 | null | B:4 |

Finally, any itemset which does not satisfy the minimum support count is discarded. The itemsets in Table 7.3 contains six itemsets that do not satisfy the 25% support threshold. Discarding these itemsets results in the final frequent itemsets: D, BA, A, BC, C and B.

## 7.1.2 FPGA Implementations of FIM Algorithms

Unfortunately, there is a limited amount of research focused on the implementation of FIM algorithms in FPGAs. A parallel implementation of the Apriori algorithm was first presented in [181], where the authors have used a 1-D systolic array. Each PE in the systolic array contains a support counter, a comparator along with some control logic and local memory. Transactions are streamed into the systolic array and compared with items stored in the local memory of each Processing Element (PE). The database is streamed into the systolic array multiple times to calculate the support count of each candidate. Due to the time required for scanning the database multiple times, only x4 speedup was achieved compared to the software

implementation. In [182], the same authors extended their work and developed a bitmapped CAM architecture that achieved a 25 times performance gain. In [183], the authors presented the HAPPI architecture which consists of a systolic array, a trimming filter and a hashing filter. The HAPPI aims at enhancing the performance of the Apriori when mining large databases by reducing the size of the database using a transaction trimming technique and by reducing the number of candidates using a hashing technique.

Although systolic array accelerators can enhance the performance of the Apriori compared to software implementations of the algorithm, the Apriori algorithm is found to lag behind other algorithms which do not require candidate generation and several database scans such as the FP-growth algorithm. Mapping the FP-growth algorithm into hardware is much more complex compared to the Apriori as it is not a simple iterative process that can be performed using the same hardware. Accelerating the algorithm is possible by performing some of its processing stages in dedicated hardware accelerators. In ([184], [185] and [63]), Sun et al. have proposed a 2-D systolic tree structure to accelerate the creation of an FP-tree of a transaction database. The systolic tree consists of several PEs each capable of storing one item of the database and representing a node of the FP-tree. A PE can be a 'parent' PE of several 'children' PEs. Children PEs associated with the same parent PE are referred to as 'siblings'. The PEs are connected together as shown in Figure 7.2, in which a parent PE is only connected to the first child and each sibling PE is connected to its neighbouring PE.

The operation of the proposed systolic tree starts by streaming the database to create the FP-tree. The items of each transaction propagate though the PEs either vertically or horizontally, so that similar itemsets always takes the same path through the PEs. This way a support count for the nodes in each path can be calculated. The next stage involves streaming candidate into the tree. At this stage the PEs execute different algorithm, so that the support count of each node at the end of each possible path for the candidate is shifted horizontally out of the tree where they are added using dedicated adders. This procedure is repeated to calculate the support count for all the candidates.

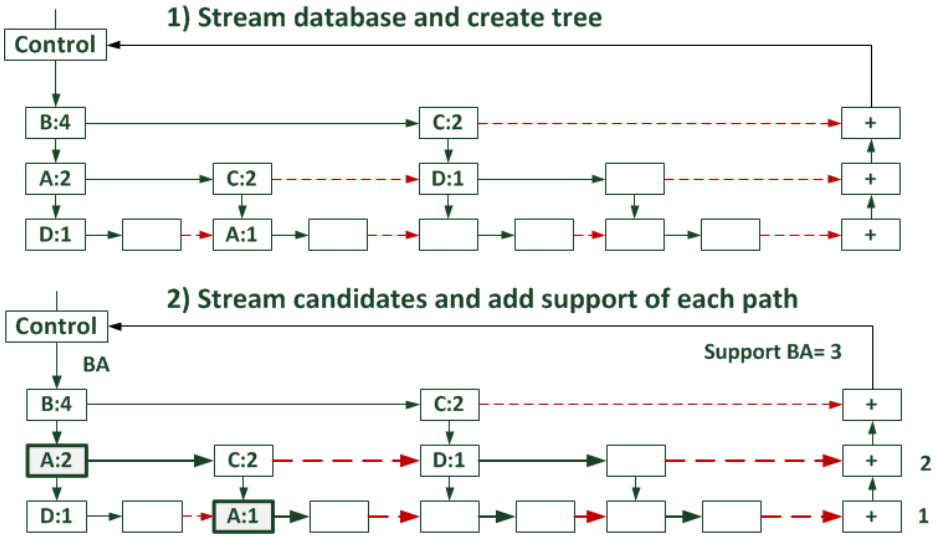



**Figure 7.2 Previously proposed 2-D systolic tree for FIM [63]**

Although the proposed tree achieved orders of magnitude performance gain over the software implementation of the FP-growth algorithm when using high support threshold, the performance gain drops dramatically when the threshold support is decreased to the point where the software implementation no longer lags behind. This is mainly due to the following two factors:

1) The PE layout in the tree requires a large number of PEs to mine a small number of items. The authors predefine a tree with two parameters K and W where K is the number of siblings PEs for each parent node and W is the depth of the tree. The total number of PEs in a tree = $(K^{w+1} -1)/(K-1)$. This means that the number of PEs is exponentially related to the K and W parameters. The authors reported that the largest tree that can fit on a Virtex-5 LX330 has K=4 and W=4. Although these parameters generate a very large tree, only transactions containing no more than four items are guaranteed to fit in the tree. To cope with this limitation, the authors proposed using a software database partitioning technique, which partitions the database into smaller sub-databases containing a maximum of four items. Only these sub-databases are mined in hardware rather than the whole database.

2) The mining process requires candidate generation, which violates one of the most important properties of the FP-growth algorithm. With database partitioning, the candidate generation process is performed after loading each sub-database.

## 7.2 Overview of Proposed System

The proposed system aims at accelerating the FP-growth algorithm through five acceleration tasks that are performed in hardware using customised systolic arrays. Table 7.4 summarises the acceleration tasks used in the proposed system. The first four tasks utilises systolic array with a conventional 1-D arrangement of PEs, whereas the fifth task utilises a systolic tree with a 2-D arrangement of PEs.

**Table 7.4 Summary of acceleration tasks in the proposed system**

| Hardware Acceleration Task | Operation |
|---|---|
| ***Item Support Counting*** | The support count is calculated for individual items in the database |
| ***Item Sorting*** | The items are sorted in a list according to their support count |
| ***Database Pruning*** | Infrequent items are removed from the database |
| ***Transaction Sorting*** | Items in transactions are rearranged in descending order of the support count |
| ***Itemset Counting*** | The support count is calculated for itemsets in a sub-database |

A vertical format is used for storing a database in memory, as shown in Figure 7.3. In this format, items in the transactions are listed using 32-bit memory locations. Each memory location is divided into two 16-bit fields. The first is used for storing the item ID and the second is used as a count field for storing some computational values during the mining process. Transactions are listed consecutively and separated by a memory location with a special ID denoted as the ‘separator’. The separator contains the value ‘0xFFFF’ in the item’s field. This format is especially designed to enable fast streaming of the database from an external DDR memory to the accelerators in the FPGA.

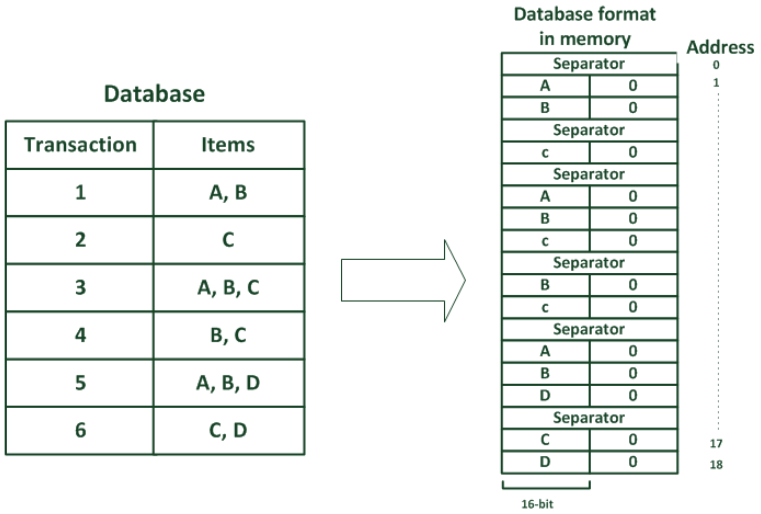



**Figure 7.3 Database format in memory**

### 7.2.1 Acceleration Task1: Item Support Counting

The initial stage when mining any database is to find the support count of individual items in the database. By using a 1-D systolic array, the database items can be counted as they are streamed into the array. The proposed 1-D systolic array consists of a number of PEs, each contains some control logic and registers to store the item ID and the support count (see Figure 7.4). PEs in the systolic array execute different algorithms according to the 'mode' signal passed to them by the neighboring PE. The systolic array used for this task operates under two modes: the LOAD mode and the SHIFT mode. The support of each item is calculated in the LOAD mode whereby each PE executes the algorithm shown in Algorithm 7.1.

In the LOAD mode the database is streamed into the systolic array. Initially, all the PEs are marked as 'empty'. Each PE contains a small ALU that performs one of three operations every clock cycle, depending on the current state of the PE and the item passed to that PE:

1) When an item reaches an empty PE, the PE is marked as 'non-empty'. The item ID is stored and the support count is set to 1. The item stopped at this PE and does not propagate to the next PE.

2) When the item reaches a non-empty PE and the item is the same as the item stored in the item ID register, the support count is incremented and the item does not propagate to the next PE.
3) When the item reaches a non-empty PE and the item is not the same as the item stored in the item ID register, the next PE is enabled and the item is passed to it.

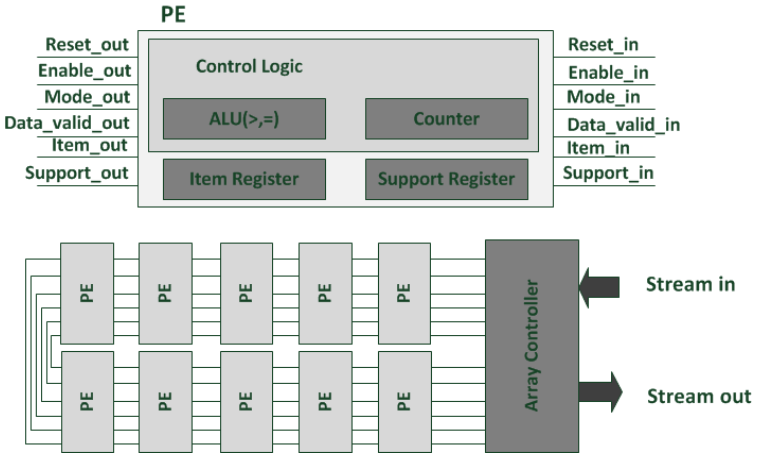



**Figure 7.4 1-D systolic array**

(1) **if** PE is empty **then**
*store item in PE*;
*support count* =1;
*stop item propagation*;
*empty_flag=1*;
(2) **else if** item= item stored in PE **then**
*support count= support count+1;*
*stop item propagation;*
(3) **else**
*forward item to next PE;*

**Algorithm 7.1 Item support counting**

If the number of PEs in the systolic array is equal or larger than the number of items in the database, the database is only required to be streamed once to finish the item counting. When streaming the database is finished, each non-empty PE will contain an item ID and a support count.

Switching the systolic array to the SHIFT mode allows for shifting the item ID and the support count of each non-empty PE out of the array. In the SHIFT mode the array behaves like a shift register, in which each PE forwards the item ID and the support count stored in its registers to the next PE and stores the incoming item ID and support count. In addition, the empty_flag in each PE is also shifted out of the array through the data_valid connection. This assists the array controller in collecting the items from the last PE.

It is not reasonable to assume that the number of PEs within the array is always larger than the number of items in the database, especially when processing a database before removing the infrequent items. When streaming a database which contains more items than the number of PEs in the systolic array, all the PEs will be filled with items and overflow will occur at the last PE in the array. To deal with this issue, the array operates as follows:

1) Any item passed by the last PE in the array to the controller during the LOAD mode is stored in memory. At the end of the first database scan a sub-database is created in memory which contains the items collected from the last PE in the array. This sub-database contains N number of items where N=total number of items-number of PEs in the array.
2) The first set of items is shifted out of the array, the array is reset and the sub-database is streamed into the systolic array rather than the full database to calculate the support of the second set of items.
3) The process of streaming the sub-database and shifting the items out of the tree is repeated. Every iteration a smaller sub-database is created until no overflow occurs at the last PE of the array.

### 7.2.2 Acceleration Task2: Item Sorting

After completing Task1, the support counts of all the items in the database are calculated. The systolic array of Task2 can be used to sort the items in a single list in a descending order of the support count. Task2 deploys a similar systolic array

architecture to that used in Task1; however, PEs execute different algorithm in the LOAD mode.

In Task2, the items along with their support counts are streamed into the systolic array. PEs perform Algorithm7.2 in the LOAD mode to sort the order of the streamed items.

(1) **if** *PE is empty* **then**
*store incoming item in PE;*
*store incoming support count in PE;*
*empty_flag=1;*
(2) **else if** *incoming support count > support stored in PE* **then**
*forward incoming item to next PE;*
*forward incoming item support to next PE;*
(3) **else**
*forward item in PE to next PE;*
*forward support in PE to next PE;*
*store incoming item in PE;*
*store incoming support in PE;*

**Algorithm 7.2 Item list sorting algorithm**

From Algorithm 7.2 it can be seen that items with the highest support counts will be shifted in the LOAD phase from one PE to the next. If the total number of items in the items group is larger than the number of PEs in the array, items with high support count will overflow at the last PE. In a similar way to Task1, these items can be collected and stored in memory to be processed in consecutive iterations.

In the SHIFT mode, the sorted items are shifted out of the array along with their support counts. As the items are shifted out of the array, the array controller examines the support count of each item. Any item with support count less than the minimum threshold is discarded and not stored in memory.

### 7.2.3 Acceleration Task3: Database Pruning

After completing Task2, a list of sorted frequent items is created. Task3 removes the infrequent items from the database to create a pruned database. In Task3 two operations are performed. The first is the removal of infrequent items from the database and the second is assigning an order number to each item entry in the database. The second operation is necessary for the following task which sorts the items in each transaction according to their support count. The systolic array in Task3 operates in two modes: the INITIALISE and the LOAD modes. In the INITIALISE mode, the items in the frequent item list are streamed into the array, each with an order number indicating the order of this item in the list (see Algorithm 7.3). The order number is set to '1' for the item with the lowest support count and increments so that the item with the highest support count has the highest order number.

```
(1) if PE is empty then
      store incoming item in PE;
      store incoming order number in PE;
      empty_flag=1;
(2) else
      forward incoming item to next PE;
      forward incoming order number to next PE;
```

**Algorithm 7.3 Initialising PEs with frequent items**

In the LOAD mode, the database is streamed into the array with the count field set to '0' for all the items. Each non-empty PE in the array compares the incoming item with the item stored in item register. If they are the same the PE forward the order number stored in the support register to the next PE (see Algorithm 7.4). This can be seen as a process of inserting an order number into the count filed of each item in the database.

Similar to the previous stages, several database scans might be required to assign all the database item entries with order numbers. The pruning process of infrequent

items is performed by the array controller in the last scan by discarding any item with no assigned order number.

```
(1) if PE is empty then
      forward incoming item to next PE;
      forward incoming order number to next PE;
(2) else if incoming item = item stored in PE
      forward incoming item to next PE;
      forward order number stored in PE to next PE;
(3) else
      forward incoming item to next PE;
      forward incoming order number to next PE;
```

**Algorithm 7.4 Assigning order numbers to items entries in the database**

### 7.2.4 Acceleration Task4: Sorting Database Transactions

After completing Task3, each memory location of the database stored in memory will contain an item ID, as well as an order number. Task4 rearranges the items in each transaction according to the order number as the database is streamed into the systolic array. The systolic array used for this task operates in two modes: the LOAD and the SHIFT modes. Different to the arrays used in the previous tasks, both the LOAD and SHIFT algorithms are executed while the database is streamed into the systolic array. In the LOAD mode, each empty PE stores the first incoming item along with its order number. Any empty PE will always forward a SHIFT mode signal to the next PE in the array. Non-empty PEs compares the order number of the incoming item with the count stored in their registers and forward the item with the highest order number to the next PE (see Algorithm 7.5). The mode signal passed to the first PE by the array controller is set to the LOAD if the item passed to the array is not the ‘separator’. When the separator is passed to the array the mode signal is switched to the SHIFT mode and the data_valid signal is set to logic ‘0’ for one clock cycle. Any PE containing the ‘separator’ will be marked as ‘empty’ during the SHIFT mode. This way the PE containing the separator will always push out the sorted items from the previous transaction.

**LOAD Mode:**

(1) **if** *PE is empty* **then**

*store incoming item in item register;*
*store incoming order number in support register;*
*forward item stored in PE to next PE;*
*forward order number stored in PE to next PE;*
*empty_flag=1;*
*mode_out= SHIFT;*
*data_valid_out=0;*

(2) **else if** *incoming item < item stored in PE*

*store incoming item in item register;*
*store incoming order number in support register;*
*forward item stored in PE to next PE;*
*forward order number stored in PE to next PE;*
*mode_out= mode_in;*
*data_valid_out=data_valid_in;*

(3) **else**

*forward incoming item to next PE;*
*forward incoming order number to next PE;*
*mode_out=mode_in;*
*data_valid_out=data_valid_in*

**SHIFT Mode:**

*store incoming item in item register;*
*store incoming order number in support register;*
*forward item stored in PE to next PE;*
*forward order number stored in PE to next PE;*
*empty_flag= data_valid_in;*
*mode_out= SHIFT;*
*data_valid_out=empty_flag;*

**Algorithm 7.5 Sorting items in database transactions**

When the size of a single transaction is larger than the size of the array, overflow will occur at the last PE. The systolic array for this task does not allow for overflow. This means that the maximum size of any transaction in the database should not exceed the number of PEs in the array. Since pruning the database transactions takes place in the previous task in the mining process, the effect of this limitation is reduced, especially when large support thresholds are selected for the mining process.

### 7.2.5 Acceleration Task5: Itemset Counting

In the FP-growth algorithm, a compressed tree of the database is created to simplify the itemset counting process. As mentioned earlier, a 2-D systolic array has been previously proposed to create the database tree in hardware (see Figure 7.2). This array process unsorted transactions, so different tree structures can be created for the same database, depending on the order of items in each transaction. This leads to inefficient mapping between the items in the database and array PEs and limits the number of items that can be mined using the array.

In this thesis, a 2-D systolic array is also proposed for creating the database tree. However, the array pre-processes the database after the completion of the first four acceleration tasks. Pre-processing the database allows for creating much larger trees with less number of PEs compared to the previously proposed array architecture.

In the proposed tree architecture, each PE is connected to two other PEs. Depending on the location of each PE within the tree, the PE can have a number of sibling PEs (connected horizontally) and children PEs (connected vertically). To simplify the connections between parent and children PEs, a parent PE is only connected to the first child, which is connected to the next sibling and so on. Figure 7.5 shows a systolic tree which supports three frequent items. The tree can be divided into several levels when moving away from the top/left PE. PEs in the same level are used for the same item. Items are assigned to the tree levels according to their order number so that the item with the highest order number is assigned to 'level 0' and the item with the lowest order number is assigned to the last level in the tree.

Each PE in the same level of the tree contains the same number of children. The number of children PEs is decremented when moving to higher levels of the tree, so that PEs in the last level are left with no children. With this array structure, the number of PEs required for the support calculation of itemsets generated from 'n' frequent items is equal to $(2^n - 1)$. This is a much smaller number of PEs compared to the previously proposed array, which has a fixed number of children for each parent PE and require a number of PEs equal to $((n^{n+1} - 1)/(n-1))$.

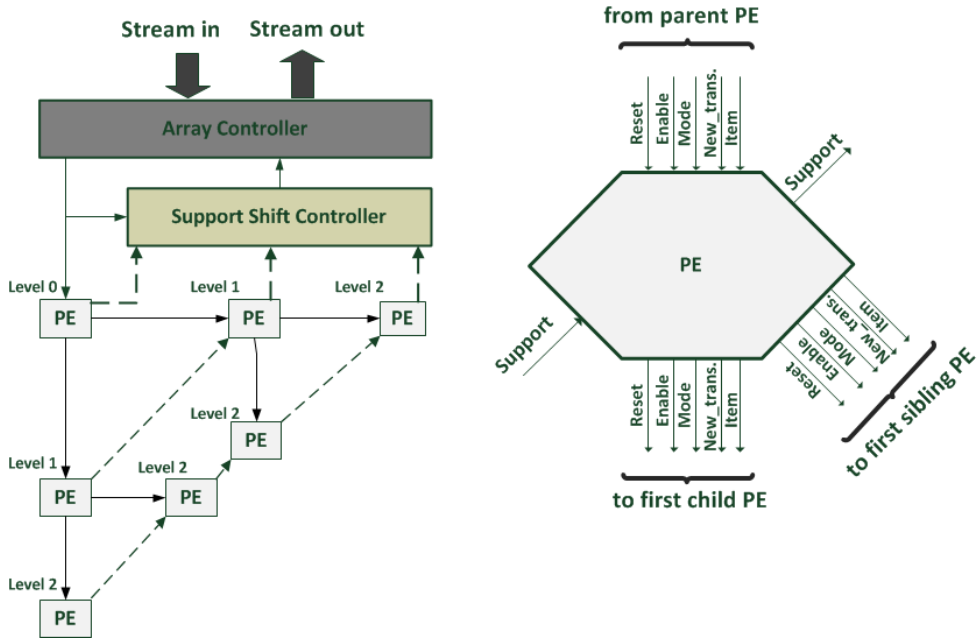



**Figure 7.5 3-item 2-D systolic array**

Similar to the 1-D systolic array used for the previous tasks, each PE in the 2-D array contains an item ID register and a support register. The proposed PE arrangement requires initialising PEs in the same level with the same item before streaming the database. The 2D systolic tree operates in three modes: the INITIALISE, LOAD and SHIFT modes.

In the INITIALISE mode, PEs are initialised with items from the sorted frequent item list by performing the algorithm described in Algorithm 7.6. Each PE stores the first incoming item in the item register. Any consecutive item will be forwarded to the first child and first sibling PEs. This way, each parent PE will have a sibling and a child PE containing the same item.

```
(1) if PE is empty then
    store incoming item in PE;
    stop item propagation
    empty_Flag=1;
(2) else
    forward incoming item to sibling PE;
    forward incoming item to child PE;
```

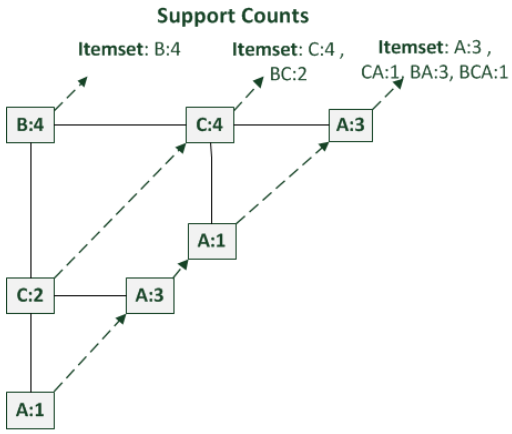
**Algorithm 7.6 Initialising the 2-D systolic array**

After streaming the frequent items into the systolic array, the mode signal is switched to the LOAD mode and the database transactions are streamed into the array. In the LOAD mode, each PE executes Algorithm 7.7 to calculate the support count of the itemsets generated from the initialised items. When performing Algorithm 7.7, items in each transaction will propagate into the tree in one or several paths. The support count of PEs in the same path will be incremented when this path is crossed by a new transaction. When streaming the database, a signal called the 'new_trans' is set for one clock cycle before each transaction. The new_trans signal is used to clear the previously created path in the tree.

```
(1) if new_trans=1 then
    path_flag=0;
    forward new_trans signal to sibling PE;
    forward new_trans signal to child PE;
(2) else if path_flag=1 then
    forward incoming item to the child PE;
    forward incoming item to sibling PE;
(3) else if incoming item= item stored in PE then
    increment support register;
    forward incoming item to sibling PE;
    path_flag=1;
(4) else
    forward item_in to sibiling PE;
```

**Algorithm 7.7 Calculating the support count of itemsets**

From Algorithm 7.7, it can be seen that each PE will always forward any incoming item to its sibling PE. This will create several independent trees within the systolic array. Figure 7.6 shows the trees created in the array when streaming an example database into a 3-item 2-D array.




Sorted Database

| Transaction | Items |
|---|---|
| 1 | B, A |
| 2 | C |
| 3 | B, C, A |
| 4 | B, C |
| 5 | B, A |
| 6 | C |

**Figure 7.6 Itemset counting using the proposed 2-D systolic array**

Shifting the calculated support counts out of the 2-D array differs from the 1-D array. PEs within the same level are connected together to form a 'shift path'. To shift items out of a particular level in the array, the array controller sets the item input signal with the ID of the item in the desired level. After that the mode signal is set to the SHIFT mode. In the SHIFT mode, the PEs perform Algorithm 7.8 to shift the support counts out of the desired level using the dedicated shift paths.

(1) **if** *incoming item = item stored in PE* **then**
*forward stored support count to next PE in same level;*
*store incoming support count in support register;*
*stop item propagation;*
(2) **else**
*forward incoming item to sibling PE;*
*forward incoming item to child PE;*

**Algorithm 7.8 Shifting items in the same level out of the 2-D systolic array**

Because there are different paths for shifting the support counts, a small 'shift controller' is attached to the array. The shift controller multiplexes the support paths to the input signal of the array controller according to the item ID set by the array controller (see Figure 7.5). In most mining problems, the size of the 2-D array will not be sufficient to mine all the items in a single database scan. There are two methods for using the 2-D systolic array in accelerating the mining process. In the first method a software database partitioning algorithm is used to create several sub-databases each not containing a number of items larger than the size of the tree [63]. Only these sub-databases are streamed into the systolic array rather than the whole database. The second method is based on database sampling, whereby the support counts of many item samples are generated in hardware. Each sample can be used to derive the association rules of a group of particular items of interest.

## 7.3 Implementation and Resource Utilisation

Tasks 1,2, 3 and 4 of the mining process are all performed using 1-D systolic arrays. It is possible to use a dedicated systolic array for each task or perform all the tasks using a single array with larger PEs capable of performing the algorithms required for the four tasks. Table 7.5 shows the resource utilisation of the 1-D systolic array when optimised for one of the four mining tasks and when optimised for all the four processing stages in a Xilinx Virtex-6 LX240 FPGA. The systolic arrays only consumes CLB resources, so the resource utilisation is given as a percentage of the total CLB slices in the FPGA which is 37,680.

**Table 7.5 CLB resource utilisation of the 1-D array in a Virtex-6 LX270 FPGA**

| Task | No. of PEs | | | |
|---|---|---|---|---|
| | **200** | **450** | **700** | **950** |
| ***Task 1*** | 12 (%) | 25 (%) | 38 (%) | 51 (%) |
| ***Task 2*** | 12 (%) | 24 (%) | 37 (%) | 49 (%) |
| ***Task 3*** | 8 (%) | 17 (%) | 26 (%) | 35 (%) |
| ***Task 4*** | 13 (%) | 27 (%) | 42 (%) | 56 (%) |
| ***All Tasks*** | 29 (%) | 59 (%) | 93 (%) | 124 (%) |

The Maximum operating frequency varies for each implementation of the 1-D systolic array. Table 7.6 shows the maximum operating frequency for the different 1-D arrays with 950 PEs.

**Table 7.6 Maximum operating frequencies for the 1-D array**

| **Array Type** | ***Task 1*** | ***Task 2*** | ***Task 3*** | ***Task 4*** | ***All Tasks*** |
|---|---|---|---|---|---|
| **Max. Frequency (MHz)** | 408 | 403 | 421 | 419 | 288 |

Task 5 of the mining process uses the 2-D systolic array. While the number of items that can be mined with the 1-D systolic array in a single database visit is equal to the number of PEs in the array, the 2-D array requires a 'PE level' for each item. This means that the size of the array almost doubles to accommodate an extra item. Table 7.7 shows the resource utilisation of the 2-D array when built to support mining a different number of items in a single database visit.

**Table 7.7 Resource utilisation of the 2-D array in a Virtex-6 LX270 FPGA**

| ***Size (No. of items)*** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|
| ***Size (No. of PEs)*** | **31** | **63** | **127** | **255** | **511** |
| ***Resource Utilisation (%)*** | 1 | 2 | 4 | 10 | 22 |
| ***Max. Frequency (MHz)*** | 360 | 356 | 327 | 290 | 290 |

To test the scalability of the proposed mining system, two designs were implemented in a Virtex-6 LX240 FPGA. In the first design, all the components of the system are static including the systolic array accelerators. A single 1-D systolic array is used for Tasks 1,2,3 and 4 of the mining process to achieve minimal area occupation. Both the 1-D array and the 2-D array share the area designated for the accelerators (see Figure 7.7). A Microblaze processor is used to control the mining tasks. Any test database in initially stored in an external CompactFlash memory. After power up of the system, the database is transferred to a DDR memory through the PLB bus which connects to a single port of an MPMC DDR memory controller. Through a Central DMA IP, the Microblaze processor can access the database stored in the DDR memory. On the other hand, streaming the database to/from the systolic array

accelerators is performed using a dedicated NPI memory controller. The NPI controller utilises two ports of the MPMC controller; one is used for reading and the other is used for writing. A single array controller is used to control the operation of the two systolic arrays according to the task and mode initiated by the Microblaze processor.

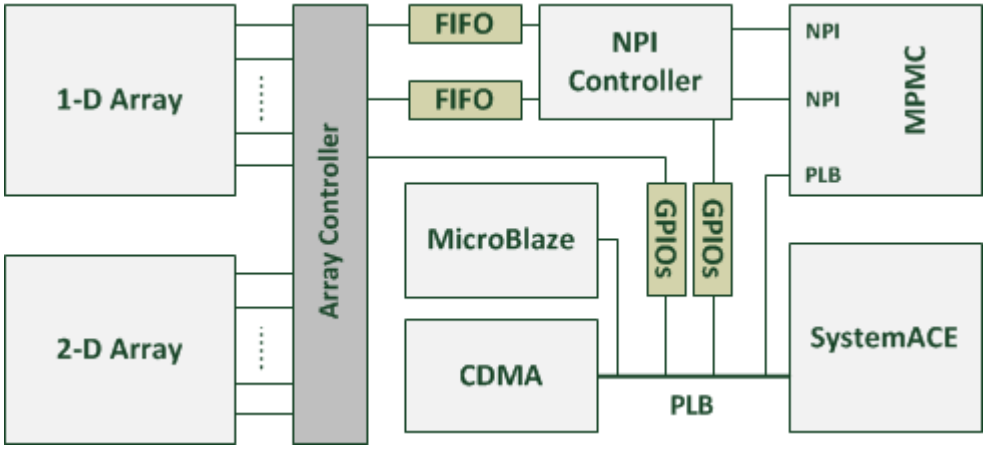



**Figure 7.7 The Static implementation of the system**

The second implemented design is partially reconfigurable. In the second design, the area designated for the accelerators is divided into different reconfigurable slots. The height of each slot is equal to height of a clock region. BMs are placed at the edges between the slots and the static region following the GoAhead design flow (see Figure 7.8) [37]. The output connections of each slot are connected to the input connections of the next slot. All the slot's output connections are also connected to multiplexers. To configure a 1-D systolic array, one or more slots can be configured with fixed-sized 1-D arrays, depending on the required overall size of the array. The array controller always streams the database data through the input connections of the first slot. On the other hand, the accelerators output data is passed to the array controller through the multiplexers. The Microblaze selects one of the multiplexers output depending on the size selected for the acceleration task. It is noted that concatenating different slots to parametrize the size of the accelerator is only applicable to the 1-D systolic arrays. The 2-D systolic array has a fixed size and is

configured on top of all the reconfigurable slots. The smallest version of the ICM presented in Chapter 4 is used to configure the accelerators using partial bitstreams stored in the DDR memory. The ICM is connected to a dedicated NPI port of the MPMC controller. The used version of the ICM does not support the multiple-clone configuration technique, which greatly reduces the configuration time for the 1-D systolic arrays. Since the locations of the slots are fixed, cloning several instances of the 1-D systolic arrays is still possible by generating several compressed partial bitstreams offline. In the implemented design, six reconfigurable slots are used for the configuration of the accelerators. This means that cloning the fixed-sized 1-D array over any number of slots require generating six compressed partial bitstreams.

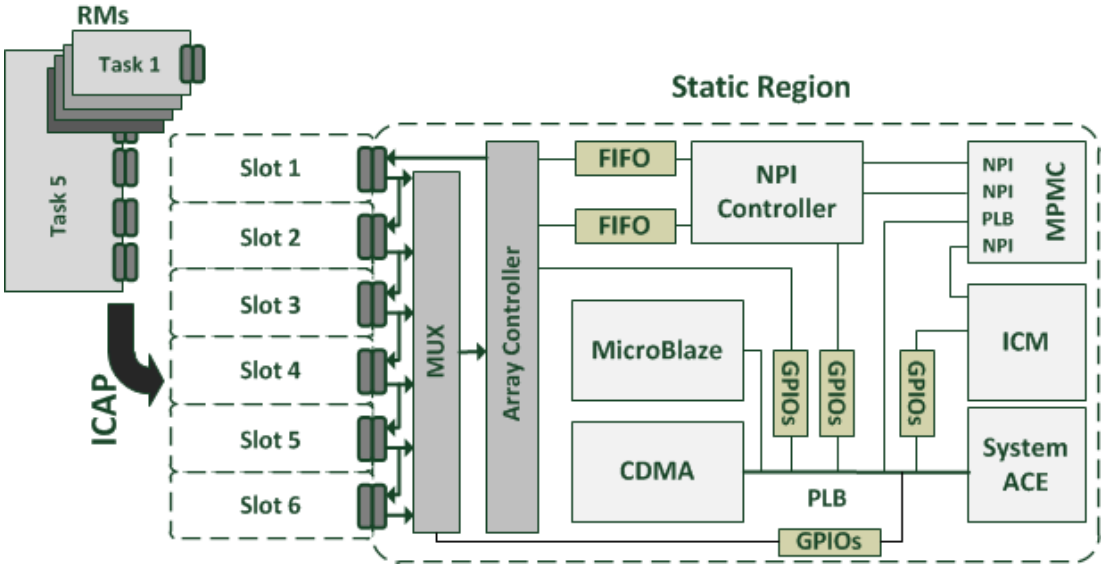



**Figure 7.8 The DPR-based implementation of the system**

In the two implementations of the system, the area designated for the accelerators is limited to around 40% of the FPGA. This area has to be shared between the 1-D systolic array and the 2-D systolic array in the static implementations. Reducing the size of the 1-D systolic array can impact the system as the number of PEs in Task4 should be larger than the maximum number of items in any database transaction. This means that the size of 1-D array should not be smaller than a specified. In this analysis, the number of PEs in the static 1-D systolic array is selected to be 200,

which consumes around 29% of the FPGA resources. This leaves around 10% of the designated area for the 2-D array (eight PE levels).

Using DPR, the systolic arrays can share the same FPGA resources within the designated area, and therefore, a larger number of PEs can fit in the designated area in each task. The designated area for the systolic array was divided vertically into six equally-sized slots. Each slot can fit up to 70 PEs of the largest 1-D systolic array (Task4). A fixed number of 70 PEs was selected for all the task accelerators, giving a total of 420 PEs when using all the reconfigurable slots. On the other hand, a 2-D array containing nine PE levels was able to fit in the designated area of the partially reconfigurable implementation. Figure 7.9 shows two floorplan images of the two prototype implementations. Table 7.8 reports the resource utilisation of the static logic in the two implementations, whereas Table 7.9 summarises the main characteristics of the two implementations of the system.

**Table 7.8 Static logic resource utilisation in a Virtex-6 LX270**

| Implementation | Static | DPR-based |
|---|---|---|
| **Slice Utilisation (%)** | 49 | 10 |
| **BRAM Utilisation (%)** | 7 | 7 |

**Table 7.9 Comparison between the two system implementations**

| Characteristics | Static | DPR-based |
|---|---|---|
| ***1-D systolic array size (PEs)*** | 200 | 420 (70 per slot) |
| ***2-D systolic array size (PEs)*** | 255 (8 items) | 511 (9 items) |
| ***Type of array*** | Static | Partially reconfigurable |
| ***Reconfiguration method*** | NA | Slot-based |
| ***1-D array reconfiguration time (ms)*** | NA | Minimum (1 slot) = 1.5<br>Maximum (6 slots) = 3.8 |
| ***2-D array reconfiguration time (ms)*** | NA | 9.2 |

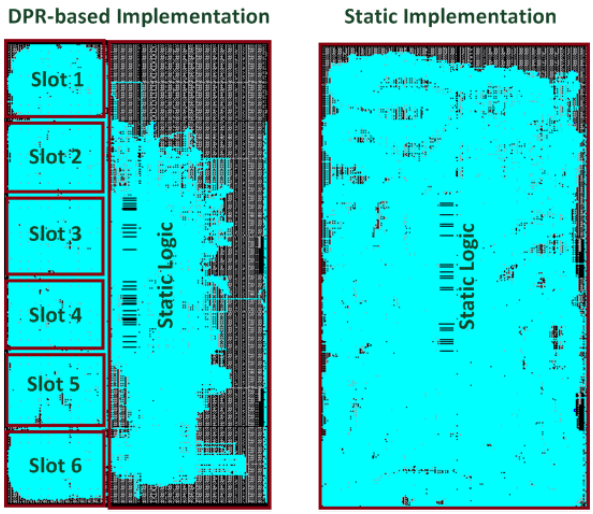



**Figure 7.9 Floorplan images of the two implementations in a Virtex-6 LX270**

## 7.4 Experimental Results

In order to analyze the performance of the proposed frequent itemset mining system, several real databases were considered as benchmarks for the two implementations. Table 7.10 lists the benchmark databases, which were collected from [186]. Most of the system's components were run at 100MHz, apart from the MPMC memory controller which takes a 200MHz reference clock. The time overheads for the different tasks of the mining process are reported in this section of the thesis. The time overhead of each task is only considered after databases are transferred to the DDR memory. The time for moving the databases from the CompactFlash memory to the DDR memory is not considered in the analysis

**Table 7.10 Benchmark databases [186]**

| Database | No. of Items | No. of Trans. | Max. Trans. Size | Min. Trans. Size | Total Items | Size in Memory (MB) |
|---|---|---|---|---|---|---|
| ***chess*** | 75 | 3196 | 37 | 37 | 118252 | 0.5 |
| ***Pumsb_star*** | 2088 | 49046 | 63 | 49 | 2475947 | 9.6 |
| ***pumsb*** | 2113 | 49046 | 74 | 74 | 3629404 | 14 |
| ***kosarak*** | 41270 | 990002 | 2498 | 1 | 8019015 | 34.4 |

### 7.4.1 Item Counting

In the first acceleration task, the database is streamed into the 1-D systolic array from the external DDR memory module to calculate the support count of the individual items. The number of times the database is streamed into the systolic array depends on the number of items in the database and the size of the array. In the static implementation, the systolic array is configured with the static components after power-up of the device, whereas for the other implementation the reconfiguration time of the systolic array account for some of the time overhead of this stage. Table 7.11 shows the time overhead for the item support counting task.

**Table 7.11 Item counting time overhead**

| Database | Time Overhead (ms) | |
|---|---|---|
| | Static | DPR-based |
| ***chess*** | 2.4 | 5.9 |
| ***pumsb_star*** | 68.0 | 58.2 |
| ***pumsb*** | 91.8 | 81.0 |
| ***kosarak*** | 2579.1 | 1213.6 |

From Table 7.11, it can be seen that the static implementation performed better in the first benchmark compared to the DPR-based implementation. This is because the benchmark database is small and only requires a single database scan to calculate the support counts of all the items. This means that the reconfiguration time of the two slots required for first task accounts for the delay in the DPR-based implementation. The DPR-based implementation; however, performed better in the other three databases because the static implementation requires more database scans to finish the task compared to the DPR-based implementation.

### 7.4.2 Sorting the Frequent Items

In this task a list of frequent items is created and sorted in descending order of the support count. In this stage, only the items collected from the previous task are streamed into the systolic array. Table 7.12 shows the time overheads for this stage.

**Table 7.12 Item sorting time overhead**

| Database | Time Overhead (ms) | |
|---|---|---|
| | Static | DPR-based |
| ***chess*** | <0.1 | 1.9 |
| ***pumsb_star*** | 0.3 | 4.1 |
| ***pumsb*** | 0.3 | 4.1 |
| ***kosarak*** | 83.8 | 41.4 |

It can be seen from Table 7.12 that in the first three databases the time overhead is dictated by the reconfiguration times in the DPR-based implementation resulting in larger time overheads compared to the static implementation. This larger overhead is because the number of items in these databases is relatively small. In the last database where the number of items is large, the DPR-based implementation performed better than the static design.

### 7.4.3 Database Pruning

In this task, each item entry in the database is assigned with an order number. This task requires the database to be streamed into the systolic array multiple times. The number of times the database is streamed into the systolic array depends on the number of items in the database, the size of the systolic array and the support threshold set for the mining problem. Figure 7.10 shows the time overheads of this task for the two implementations when varying the support threshold.

From Figure 7.10, it can be seen that the higher the support threshold, the smaller the time overhead of this task. The time overhead depends on the number of items in the

frequent item list. Increasing the support threshold decreases the number of frequent items and consequently decreases the number of database scans in this task. When the number of frequent items is smaller than the number of PEs in the array, decreasing the support count does not affect the time overhead of this task.

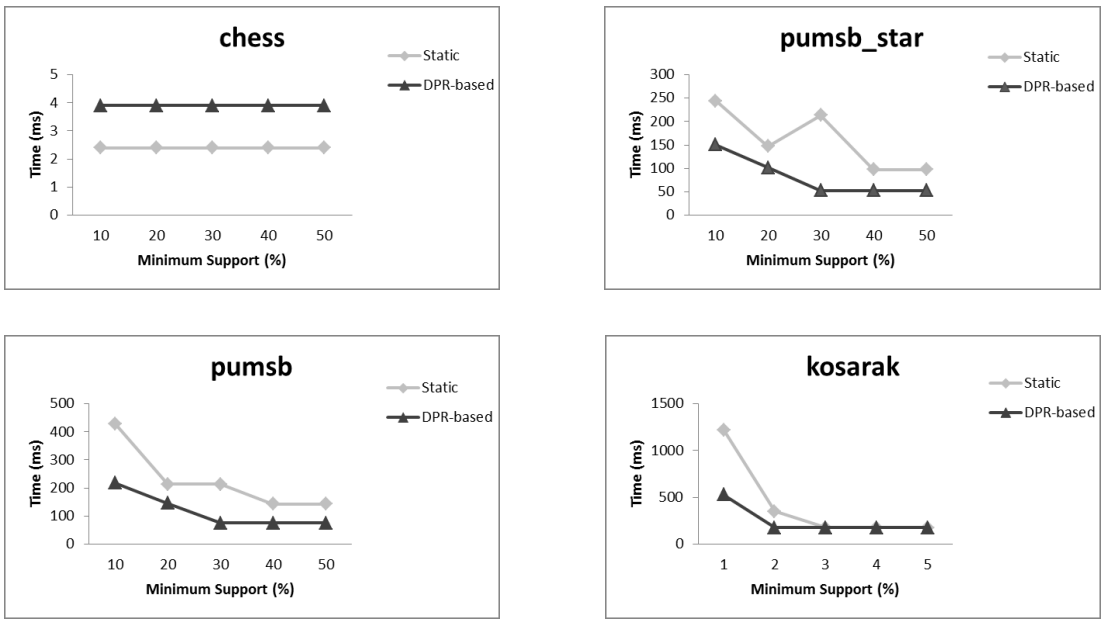



**Figure 7.10 Database pruning time overhead**

## 7.4.4 Sorting Database Transactions

This task requires the pruned database created in the previous task to be streamed once into the systolic array to rearrange the frequent items in each transaction. Array overflow is not considered in this task. This poses a minor limitation as the maximum number of items within any transaction must not exceed the array size. Experimental analysis of the benchmark databases showed that the first three databases in Table 7.10 do not contain any transaction bigger than 200 which is the size of the smallest array in the static design. For the last database, setting the support threshold in the previous tasks to 1% will guarantee that no transaction contains more than 200 items. Table 7.13 shows the database sorting time when setting the lowest

possible support count for the static implementation. It is noted that the maximum number of slots is configured in the DPR-based design stage which results in a small lag in the execution time of this task.

**Table 7.13 Time overhead for sorting database transactions**

| Database | Minimum Support | Time overhead (ms) | |
|---|---|---|---|
| | | Static | DPR-based |
| ***chess*** | 1 | 2.4 | 6.2 |
| ***pumsb_star*** | 1 | 48.9 | 52.7 |
| ***pumsb*** | 1 | 71.3 | 75.1 |
| ***kosarak*** | 9900 | 74.7 | 78.5 |

### 7.4.5 Itemset Counting

This task is performed using the 2-D systolic array. This array can be used to accelerate the itemset counting process by generating some of the frequent itemsets in hardware. A single task calculates the support counts of the itemsets generated from the items initialised in the array. This task can be used to sample the database by generating the itemsets from particular items of interest. In the DPR-based implementation, the size of the array is one level larger compared to the array in the static implementation. Figure 7.11 shows the time overhead for generating the itemset support counts from "k" items in the benchmark databases, whereby k is the number of levels in the systolic array. The DPR-based design takes slightly more time to complete this task. This is mainly because of the reconfiguration delay. The DPR-based design however, calculates the support counts of double the amount of itemsets in every sample of the database. This means that the overall acceleration can be much higher with the DPR-based design. If we consider the FPGA-based mining system proposed in [63] which also utilises a 2-D systolic array for itemset counting, we can see that the number of support counts generated in every sample is much smaller compared to the system proposed in this thesis. In addition, the time overhead of database sampling in the system proposed in [63] is much larger due to applying 'candidate generation' for each sample. To demonstrate the benefit of the

larger tree in the DPR-based implementation, Figure 7.12 shows a simulation of the number of support counts calculated in hardware with respect to the database sampling time for the 'chess' database, when setting the support threshold to 10%.

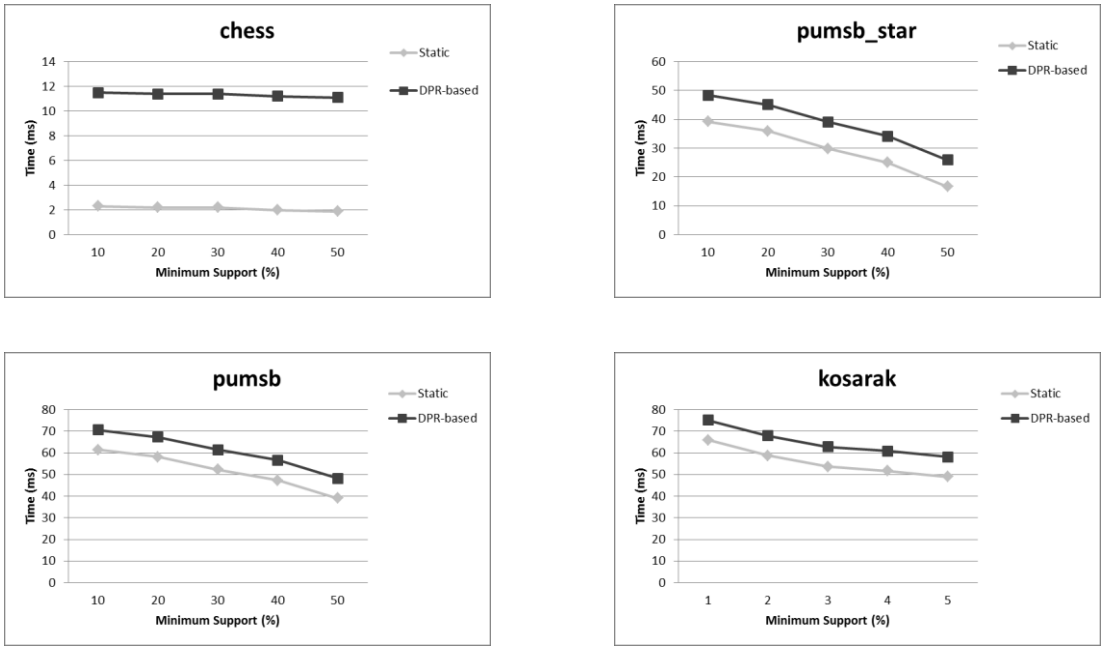



**Figure 7.11 Itemset support calculation time overhead for top-k items**

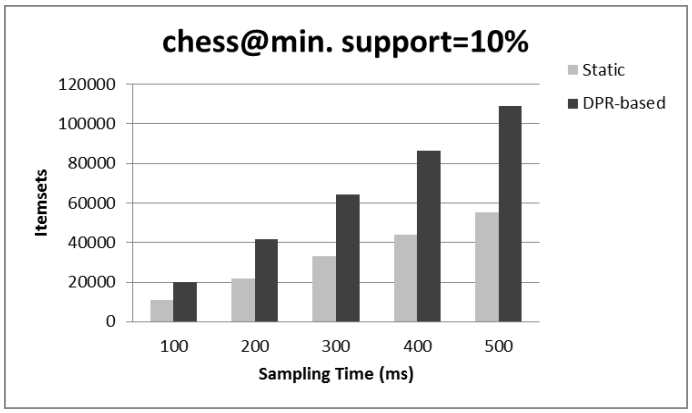



**Figure 7.12 Support counts calculated in hardware**

## 7.5 Chapter Conclusion

DPR can greatly extend the flexibility for designing software acceleration platforms in FPGAs. Usually, the available resources in FPGAs can be used to implement fixed hardware to accelerate certain portions of the software. With DPR, different accelerators can be swapped in/out of the FPGA to extend the number of tasks performed in hardware and consequently enhance the overall performance of the system. This however requires efficient management of the acceleration tasks for a given application.

This chapter presented an FPGA-based reconfigurable platform which aims to accelerate the FP-growth algorithm for FIM applications. The proposed platform divides the FP-growth algorithm into five acceleration tasks each is performed using a specialised systolic array accelerator. Four of the accelerators are based on a traditional 1-D systolic array architecture whereas one accelerator is based on a 2-D architecture. Two implementations of the proposed platform where demonstrated and evaluated. The first implementation deploys all the accelerators in a static design, wherein all the accelerators share the available FPGA resources. The second implementation was designed with a DPR design flow so that accelerators can be swapped at run time allowing for placing larger systolic arrays for each acceleration task. Using a slot-based architecture, the size of the 1-D systolic array accelerators can be parametrized by selecting the required number of reconfigurable slots hosting the accelerators. The regularity of 1-D systolic array accelerators allowed for the reconfiguration delay to be significantly reduced using the multiple clone configuration technique by cloning the required number of slots (see Chapter 4). Experimental analysis with real database benchmarks showed that the DPR-based implementation can achieve better overall acceleration in most benchmarks despite the reconfiguration overhead. The static implementation performs better than the DPR-based implementation when the database is very small and does not require larger systolic arrays, which is not the case for most real databases.

# Chapter 8 : Conclusion and Future Work

This thesis presented several innovative solutions for the internal management of DPR in SRAM FPGAs. These solutions addressed two major challenges in the field of reconfigurable computing, namely, performance and reliability. Since its introduction in the FPGA industry, DPR has been seen as an exciting opportunity to implement new solutions to enhance the reliability and performance of many applications. However, the deployment of DPR in today's real-world applications is nowhere near its full potential, despite the continuously expanding portfolio of devices supporting this feature. This is mainly due to the practical difficulties in designing DPR applications and the lack of generic design platforms that naturally support high performance and reliability.

While traditional DPR design flows deploy DPR as a method for basic context-switch operations of some modules over a defined physical space on the FPGA chip, this thesis aimed to better exploit the FPGA's resources by efficiently managing the configuration and execution of fully relocatable modules that perform specific computational tasks. This thesis addressed all the design issues and challenges in implementing a practical ROS system. Indeed, the work presented in this thesis has paved the way for the development of the Reliable Reconfigurable Real-Time Operating System (R3TOS), which aims to be a solid platform for fault-tolerant applications in reconfigurable hardware.

The reminder of this chapter summarises the research work covered in the thesis, draws conclusions and evaluates the impact of the achievements of the thesis. Finally, a discussion highlighting the key areas that require improvement is presented, along with planned future work related to this thesis.

## 8.1 Summary and Concluding Remarks

The main contributions in this thesis are presented in Chapters 4, 5, 6 and 7. Chapter 4 presented the design and architecture of an Internal Configuration Manager (ICM) that supports a wide range of configuration operations. The Xilinx Virtex FPGA family was the selected target for implementing the proposed ICM, which provides several advantages over the currently available ICAP controllers. First of all, the proposed ICM is very flexible, supporting a wide range of configuration operations that can be tailored to the needs of a particular application. For example, the ICM can operate as a stand-alone configuration memory scrubber or as a controller for basic DPR operations. The ICM also supports more advanced operations that are particularly important for implementing a practical ROS kernel. Bitstream relocation can be performed at speeds approaching the theoretical limit of the ICAP, making the proposed ICM multiple times faster than the current relocation filters. In addition, the relocation and configuration processes are entirely handled by the ICM, allowing the system to run more efficiently and to have better multi-tasking capabilities.

Chapter 4 also introduced two new features for managing ROS tasks. The first feature allows for generating a black-box bitstream online to quickly remove the configuration of a particular area on the FPGA. This feature is referred to as 'task blanking' as it can be used in an ROS to efficiently manage the removal of already configured modules that are not being used by any task. Removing unused modules can reduce the static power dissipation of the system; however, it should be performed very efficiently as extra delays incurred from the task removal process can degrade the performance of the system. Task blanking is very beneficial when implementing an ROS, as no black-box bitstream configuration is required for each relocatable module in the system and this can reduce the size of the storage memory required for the system. The second advantage of task blanking is the support of black-box bitstream compression, which greatly reduces the task removal speed and consequently enhances the performance of the system. The second novelty feature of the proposed ICM is the multiple-clone configuration technique which, allows for the configuration of multiple instances of a relocatable module at the same time. The

configuration time of this technique can be several times smaller than the conventional configuration method. This feature can be particularly useful in an ROS continuously scheduling real-time tasks for execution on the same relocatable module.

Chapter 5 discussed how flexible FT systems can be achieved with reliable configuration management. The chapter first presented a discussion of several possible design techniques to make the proposed ICM resilient to emerging faults. These techniques allow for internal faults within the ICM logic to be corrected by partial reconfiguration. The experimental analysis showed that, although modular redundancy is very effective in detecting faults in the ICM, the resource utilisation of designs based on modular redundancy can be intolerable in systems using the full capabilities of the ICM. A novel self-recovery system is proposed to reduce the resource utilisation of modular redundancy. The system consists of two RPs, one containing the full ICM and the other containing a small recovery controller designed with TMR. The full ICM is connected to the ICAP by default; however, when a fault is detected in the operation of the ICM, access to the ICAP is switched to the smaller recovery controller, which reconfigures the full ICM. Appling TMR to a small part of the system resulted in much smaller area occupation compared to full TMR and DMR of the ICM, without reducing the self-healing capabilities. Smaller area occupation for the static components is always desirable in ROS implementations as this means more free area for the execution of tasks and, in turn, leads to better performance and multi-tasking. Chapter 5 also demonstrated how the ICM can be used for fault mitigation in the rest of the FPGA's reconfigurable resources. Dealing with soft faults was first discussed and several configuration memory scrubbing techniques were tested and evaluated. The proposed ICM mainly gives support to readback scrubbing as well as external scrubbing to correct bit-flips in the FPGA's configuration memory. Using the proposed ICM, the two scrubbing methods can be used in the same system. This can increase the reliability and reduce the probability of system failure as the number of single points of failure is reduced to the points located at the interconnections between the ICM and the ICAP. The ability to utilise the full capabilities of the ICM in a reliable manor enabled the implementation of a

comprehensive fault-handling system that mitigates emerging permanent faults as well as soft faults. Chapter 5 introduced the R3TOS platform, which allows for the execution of tasks using fully isolated relocatable modules. The relocatable module architecture in R3TOS provides some important reliability features not present in any relocation system. First of all, the fact that the modules are fully isolated means that it is highly unlikely that a single fault will propagate from one module to the other. This is a problem often seen when a fault alters the routing configuration in the FPGAs, causing damage to several modules that share common routes and signals. Moreover, in ordinary relocation systems, relocation is only permitted to locations containing fixed bus-macros to allow for communicating with the relocatable module after configuration. This is not the case in R3TOS, as its ICAP-based communication scheme allows for the transfer of data from/to the relocatable modules without the need for fixed physical routes and this increases the number of feasible locations for each relocatable module. The flexible relocation in R3TOS makes the addition of redundant modules for the critical tasks less costly on the performance of the system. In fact, using three redundant modules for each relocatable core does not just guarantee correct task execution, it also simplifies permanent fault detection and greatly reduces the time of permanent fault diagnosis. When a fault is detected by task redundancy, only the region occupied by the faulty module is tested once. Considering that permanent fault diagnosis has a large time overhead, performing on-demand tests on specific regions on the FPGA is much more efficient compared to the conventional online testing schemes, which are based on periodic tests over the entire FPGA. This thesis proposes using relocatable BIST circuits that can be tiled together to test the resources in an area with any size and shape. Using relocatable BIST circuits means that only a few configurations are required to be stored in external memory. In addition, the multiple-clone configuration technique can significantly reduce the diagnosis time overhead which is the major disadvantage of online BIST diagnosis.

A case study application that greatly benefits from the proposed reliability-centric configuration management scheme was presented in Chapter 6. Chapter 6 presented the design and architecture of a flexible encryption engine over the R3TOS design

platform. The engine is designed so that encryption tasks are performed using the proposed redundancy system to ensure correct functionality. The reliability of the proposed system will guarantee that no secret information is leaked as a result of faults in the cipher cores. In fact, the system utilises all the fault detection and correction features proposed in the thesis in a single fault-handling scheme. The system also introduces a new placement method for relocatable modules consisting of multiple-resource types. The proposed placement method does not require a full scan of the FPGA's resource to identify feasible locations for the modules. The placement method reduces the FPGA's horizontal scan time by storing fixed offset groups pointing to the regions with identical resource layout. A relocatable module is assigned to one of these offset groups, which means that the horizontal locations are always pre-determined. Good partitioning of the FPGA's horizontal layout showed that only a few bytes of memory are required for all the offset groups in the largest Virtex-4 FPGA device. In addition, an intelligent module-reuse scheme is introduced to manage the configuration of relocatable modules in the system. The scheme keeps track of the already configured modules in the FPGA to enable them to be reused for future tasks. This scheme significantly enhances the performance of the system, especially when specific modules are heavily utilised during the operation of the system.

Chapter 7 presents another case study application that is focused on achieving high-performance software acceleration of database mining algorithms using relocatable systolic-array accelerators. The chapter introduced the design and architecture of a DPR-based platform to accelerate the FP-growth algorithm that is widely used in Frequent Itemset Mining (FIM). The system performs five acceleration tasks in hardware and each is executed using a specialised systolic array accelerator. While previous work utilised a single systolic array to accelerate a small portion of the algorithm, the proposed system deploys DPR to enable more acceleration by time-sharing the FPGA's resources among the different accelerators. The resource efficiency gained from DPR not only allows for accelerating more parts of the algorithm in hardware, but also allows for designing innovative high-performance accelerators that would not normally fit within the limited FPGA resources. Four of

these systolic array accelerators are designed with a standard 1-D arrangement of PEs, whereas one of the arrays is designed with a novel 2-D arrangement of PEs. The 2-D systolic array accelerates the itemset counting stage, which is the most computationally intensive task in the FP-growth algorithm. Compared to a previously reported static systolic array architecture, the proposed architecture can achieve more than double the acceleration for itemset counting. The high-speed configuration possible with the proposed ICM greatly reduces the effect of reconfiguration delay in the system performance. Moreover, the flexibility of module relocation allows for parametrising the size of the accelerators to fit the computational demands of the acceleration tasks.

## 8.2 Future Work

There are several aspects of the presented work that would benefit from further investigation and improvement. First of all, the presented ICM was mainly demonstrated using the Virtex-4 FPGA family. Although the configuration architecture and features are almost identical in the new generations of Xilinx FPGAs, the ICM cannot be directly applicable to the newer FPGAs (e.g. Xilinx 7-series FPGAs). To be more precise, some configuration operations related to bitstream relocation require the reverse engineering of portions of the target FPGA's bitstream. For example, online routing requires knowledge of the configuration bits responsible for enabling/disabling the regional clock buffers and varying the frequency of regional clock signals. In addition, ICAP-based communication requires knowledge of the LUTs configuration bits as well as the BRAM content mapping into the bitstream. Moving the technology to a newer generation of FPGAs requires repeating all the reverse engineering experiments.

All the presented prototype designs utilised a soft-processor implemented in the FPGA's logic to control the system. A soft-processor utilises a large area and this can impact the performance of the system. In the first case study, the R3TOS system deployed a Microblaze soft-processor to control the execution of encryption tasks in the FPGAs (see Chapter 6). Despite the fact that the system's main focus is the faults

that will affect the operation of the encryption cores, faults in the static control logic could cause the system to fail. The large resource utilisation of the soft-processor prevents the application of TMR in the static logic as this will greatly reduce the area dedicated for the execution of tasks and consequently degrade the performance of the system. In addition, the performance of soft-processors is significantly inferior to ASIC processors and this can be a performance bottleneck when the processor is required to perform intensive computations, as seen in the second case study in this thesis (see Chapter 7). In this case study, the modest performance of the Microblaze processor made the system unable to perform 'database partitioning' which is a highly performance-demanding operation. This means that connecting an external, more powerful processor to the FPGA is a more suitable choice for such applications.

Bitstream security is another issue facing relocation systems in general. Relocation requires modifying the original bitstream. When using encrypted bitstreams, a decryption function must be implemented in the FPGA's logic to retrieve the configuration data from the encrypted bitstream and enable the configuration manager to perform the necessary modifications for relocation. This can affect the performance of the system in two ways: first, the decryption function may delay the configuration process if not implemented in a pipelined architecture; second, the area utilisation of the cipher can reduce the area dedicated for the relocatable modules. This problem can be seen in Chapter 6 where only simple scrambling algorithm is used for bitstream encryption. This kind of encryption is definitely insufficient, especially with the increasing number of side-channel attacks targeting FPGA devices.

Finally, it is clear that current commercial FPGAs are not designed to implement systems that heavily utilize the configuration port such as the R3TOS. The maximum configuration throughput is 400 MB/s and the newer generation of FPGAs is not showing any improvement in configuration speed. This imposes some limitations on the type of application that can be implemented using R3TOS. The ICAP-based communication is the main aspect of R3TOS that would suffer from the limited ICAP speed. Many applications require high-speed streaming of data to the

relocatable cores. Implementing such applications may be difficult, especially when using modular redundancy as data needs to be transferred more than once using the ICAP. Of course, the multiple-clone configuration technique can be deployed to reduce the delay of data transfer; however, for wider deployment of this technology, the FPGA manufacturers should address the limited configuration throughput in future devices.

To address the aforementioned concerns, several directions can be taken for the future work following this thesis. The planned future work is summarised as follows:

- **Support for state-of-the-art devices**: A new breed of reconfigurable devices consisting of an ASIC processor and an FPGA fabric in a single SoC is taking off and catching the attention of researchers in the reconfigurable computing community. An example of these devices is the Xilinx Zynq-700 SoCs which consist of a dual-core ARM processor and a 7-series FPGA fabric [54]. Migrating the presented systems and designs to such platforms will address two limitations of the presented work. First, the performance of the ARM processor is much superior to the performance of the Microblaze soft-processor and this enables the implementation of more complex systems and allows for better control over the target application. Second, removing the soft-processor frees a large area from the FPGA's fabric, allowing for the integration of more specialised RMs and consequently enhancing the performance of the system. In fact, a joint collaboration between the University of Edinburgh and JPL has recently started to migrate the design of the presented ICM to the Zynq-7000 devices. The main aim of the joint project is to create a generic reliable computational platform for common space applications.

- **Solid device security**: As stated earlier in this chapter, relocating encrypted bitstreams is a major challenge requiring the implementation of a small footprint, high-speed decryption block inside the FPGA. The bitstream encryption must be strong enough and immune to common attacks targeting FPGAs to justify using the presented ICM in commercial application. There are several pipelined implementations of the AES algorithm especially optimised for FPGAs.

Removing the soft-processor from the FPGA might allow for placing a large decryption block such as the AES. Other decryption blocks with much smaller area footprints are worth investigating such as the bitstream encryption system presented in [187], which is patented by Altera.

- **High-speed IOs**: In many applications, large datasets are required to be transferred to/from the FPGA. For example, the data mining acceleration system presented in Chapter 7 requires the transfer of large portions of the database to the FPGA's main memory. To allow for such data transfers, integration of the PCIe standard is planned for the proposed system.

- **Reliability testing**: Performing fault injection tests using the configuration port of the FPGA may not be enough to justify the reliability of the system. One millstone of the joint project between the University of Edinburgh and JPL is to perform real radiation tests to better understand the effects of faults on the ICM and the reliability of the system.

- **High-level programming**: This thesis demonstrated how the R3TOS can be a generic platform for implementing FT applications. The next natural step is to develop a high-level programming tool that allows designers to code their applications entirely in software.

# References


[1] IBM, "The Evolution of the Electronics Industry," *IBM Document: ELE03005USEN,* 2012.

[2] J. Chelikowsky, "Introduction: Silicon in All Its Forms," *in Silicon: Evolution and Future of a Technology, Springer Berlin Heidelberg,* pp. 1-22, 2004.

[3] Gartner. (2014). *Market Share: All Software Markets, Worldwide, 2013 [Online]*. Available: http://www.gartner.com/newsroom/id/2696317

[4] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference,* pp. 483-485, 1967.

[5] J. Diaz, C. Munoz-Caro, and A. Nino, "A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era," *IEEE Transactions on Parallel and Distributed Systems,* vol. 23, no.8, pp. 1369-1386, 2012.

[6] U. Vishkin, "Is Multicore Hardware for General-Purpose Parallel Processing Broken?," *Communications of the ACM,* vol. 57, no.4, pp. 35-39, 2014.

[7] Xilinx, "Xilinx Ships Industry's First 20-nm All Programable Devices," *XCELL Journal, Issue 86, First Quarter* 2014.

[8] Xilinx, "Introduction to FPGA Design with Vivado High-Level Synthesis," *Xilinx Document: UG998,* 2013.

[9] Altera, "Implementing FPGA Design with the OpenCL Standard," *Altera White Paper: WP-01173-3.0,* 2013.

[10] M. Graphics, "Keep Your FPGA Options Open with Vendor-Independent IP," *Mentor Graphics White Paper: TECH8570-w,* 2009.

[11] Synopsys, "Synplify Premier: Fast, Reliable FPGA Implementation and Debug," *Synopsys Datasheet,* 2012.

[12] G. Brebner, "A Virtual Hardware Operating System for the Xilinx XC6200," *The International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers,* pp. 327–336, 1996.

[13] A. Ebrahim, K. Benkrid, X. Iturbe, and C. Hong, "A Novel High-Performance Fault-Tolerant ICAP Controller," *The NASA/ESA Conference on Adaptive Hardware and Systems (AHS),* pp. 259-263, 2012.

[14] A. Ebrahim, K. Benkrid, X. Iturbe, and C. Hong, "Multiple-Clone Configuration of Relocatable Partial Bitstreams in Xilinx Virtex FPGAs," *The NASA/ESA Conference on Adaptive Hardware and Systems (AHS),* pp. 178-183, 2013.

[15] A. Ebrahim, T. Arslan, and X. Iturbe, "On Enhancing the Reliability of Internal Configuration Controllers in FPGAs," *The NASA/ESA Conference on Adaptive Hardware and Systems (AHS),* pp. 83-88, 2014.

[16] X. Iturbe, A. Ebrahim, K. Benkrid, C. Hong, T. Arslan, J. Perez*, et al.*, "R3TOS-Based Autonomous Fault-Tolerant Systems " *IEEE Micro,* vol. 34, no.6, pp. 20-30, 2014.

[17] X. Iturbe, K. Benkrid, C. Hong, A. Ebrahim, R. Torrego, I. Martinez*, et al.*, "R3TOS: A Novel Reliable Reconfigurable Real-Time Operating System for Highly Adaptive, Efficient, and Dependable Computing on FPGAs " *IEEE Transactions on Computers,* vol. 62, no.8, pp. 1542-1556, 2013.

[18] A. Ebrahim, T. Arslan, and X. Iturbe, "A Fast and Scalable FPGA Damage Diagnostic Service for R3TOS Using BIST Cloning Technique " *The International Conference on Field Programmable Logic and Applications (FPL),* pp. 1-4, 2014.

[19] P. H. W. Leong, "Recent Trends in FPGA Architectures and Applications," *The IEEE International Symposium on Electronic Design, Test and Applications (DELTA),* pp. 137-141, 2008.

[20] J. Rose, R. Tessier, and I. Kuon, "FPGA Architecture: Survey and Challenges," *Foundations and Trends in Electronic Design Automation,* vol. 2, no.2, pp. 135-253, 2007.

[21] H. Styles and W. Luk, "Compilation and Management of Phase-Optimized Reconfigurable Systems," *The International Conference on Field Programmable Logic and Applications (FPL),* pp. 311-316, 2005.

[22] M. J. Wirthlin and B. L. Hutchings, "Improving Functional Density Using Run-Time Circuit Reconfiguration " *IEEE Transactions on Very Large Scale Integration (VLSI),* vol. 6, no.2, pp. 247-256, 1998.

[23] S. Liu, R. N. Pittman, and A. Forin, "Energy Reduction with Run-Time Partial Reconfiguration," *Technical Report of Microsoft Research: MSR-TR-2009-2017,* 2009.

[24] EE Times. (2014). *FPGAs as ASIC Alternatives: Past & Future [Online]*. Available: http://www.eetimes.com/author.asp?doc_id=1322021

[25] Xilinx, "Virtex-4 FPGA Configuration User Guide," *Xilinx Document: UG071,* 2009.

[26] P. B. Minev and V. S. Kukenska, "The Virtex-5 Routing and Logic Architecture," *Annual Journal of Electronics,* pp. 107-110, 2009.

[27] X. Iturbe, K. Benkrid, R. Torrego, A. Ebrahim, and T. Arslan, "Online Clock Routing in Xilinx FPGAs for High-Performance and Reliability," *The NASA/ESA Conference on Adaptive Hardware and Systems (AHS),* pp. 85-91, 2012.

[28] Xilinx, "PlanAhead UsercGuide," *Xilinx Document: UG632,* 2012.

[29] Xilinx, "Partial Reconfiguration User Guide," *Xilinx Document: UG702,* 2013.

[30] M. Bourgeault. (2011). *Altera Partial Reconfiguration Flow*. Available: http://www.eecg.utoronto.ca/~jayar/FPGAseminar/FPGA_Bourgeault_June23_2011.pdf

[31] D. Koch, "Partial Reconfiguration in Space and Time," *in Partial Reconfiguration on FPGAs: Architectures, Tools and Applications, Springer Science & Business Media,* pp. 29-37, 2012.

[32] A. Wold, A. Agne, and J. Torresen, "Relocatable Hardware Threads in Run-Time Reconfigurable Systems," *Reconfigurable Computing: Architectures, Tools, and Applications (ARC),* pp. 62-72, 2014.

[33] T. Becker, W. Luk, and P. Y. K. Cheung, "Enhancing Relocatability of Partial Bitstreams for Run-Time Reconfiguration," *The IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM),* pp. 35-44, 2007.

[34] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght, "Modular Dynamic Reconfiguration in Virtex FPGAs," *IEEE Proceedings - Computers and Digital Techniques,* vol. 153, no.3, p. 157, 2006.

[35] M. L. Silva and J. C. Ferreira, "Generation of Partial FPGA Configurations at Run-Time," *The International Conference on Field Programmable Logic and Applications (FPL),* pp. 367-372, 2008.

[36] A. A. Sohanghpurwala, P. Athanas, T. Frangieh, and A. Wood, "OpenPR: An Open-Source Partial-Reconfiguration Toolkit for Xilinx FPGAs," *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW),* pp. 228-235, 2011.

[37] C. Beckhoff, D. Koch, and J. Torresen, "Go Ahead: A Partial Reconfiguration Framework," *The IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM),* pp. 37-44, 2012.

[38] H. Shayani, P. Bentley, and A. M. Tyrrell, "A Cellular Structure for Online Routing of Digital Spiking Neuron Axons and Dendrites on FPGAs," *Evolvable Systems: From Biology to Hardware,* pp. 273-284, 2008.

[39] C. Claus, B. Zhang, M. Hübner, C. Schmutzler, and J. Becker, "An XDL-Based Busmacro Generator for Customizable Communication Interfaces for Dynamically and Partially Reconfigurable Systems," *Workshop on Reconfigurable Computing Education,* 2007.

[40] S. Korf, D. Cozzi, M. Koester, J. Hagemeyer, M. Porrmann, U. Rückert*, et al.*, "Automatic HDL-Based Generation of Homogeneous Hard Macros for FPGAs," *The IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM),* pp. 125-132, 2011.

[41] C. Bobda, M. Majer, A. Ahmadinia, T. Haller, A. Linarth, and J. Teich, "The Erlangen Slot Machine: Increasing Flexibility in FPGA-Based Reconfigurable Platforms," *The IEEE International Conference on Field-Programmable Technology (FPT),* pp. 37-42, 2005.

[42] L. Devaux, S. Ben Sassi, S. Pillement, D. Chillet, and D. Demigny, "Flexible Interconnection Network for Dynamically and Partially Reconfigurable Architectures," *International Journal of Reconfigurable Computing,* vol. 2010, no. 390545, pp. 1-15, 2010.

[43] D. Koch, C. Beckhoff, and J. Teich, "ReCoBus-Builder — A Novel Tool and Technique to Build Statically and Dynamically Reconfigurable Systems for FPGAS," *The International Conference on Field Programmable Logic and Applications (FPL),* pp. 119-124, 2008.

[44] Y. E. Krasteva, A. B. Jimeno, E. de la Torre, and T. Riesgo, "Straight Method for Reallocation of Complex Cores by Dynamic Reconfiguration in Virtex II FPGAs," *The IEEE International Workshop on Rapid System Prototyping,* pp. 77-83, 2005.

[45] E. Horta and J. W. Lockwood, "PARBIT: A Tool to Transform Bitfiles to Implement Partial Reconfiguration of Field Programmable Gate Arrays (FPGAs)," *Technical Report: WUCS-01-13,* 2001.

[46] Y. E. Krasteva, E. de la Torre, T. Riesgo, and D. Joly, "Virtex II FPGA Bitstream Manipulation: Application to Reconfiguration Control Systems," *The International Conference on Field Programmable Logic and Applications (FPL),* pp. 1-4, 2006.

[47] J. Carver, R. N. Pittman, and A. Forin, "Relocation and Automatic Floor-Planning of FPGA Partial Configuration Bitstreams," *Technical Report of Microsoft Research: MSR-TR-2008–111,* 2008.

[48] H. Kalte, G. Lee, M. Porrmann, and U. Ruckert, "REPLICA: A Bitstream Manipulation Filter for Module Relocation in Partial Reconfigurable Systems," *The IEEE International Symposium on Parallel and Distributed Processing,* p. 151b, 2005.

[49] H. Kalte and M. Porrmann, "REPLICA2Pro: Task Relocation by Bitstream Manipulation in Virtex-II/Pro FPGAs," *The 3rd Conference on Computing Frontiers,* pp. 403-412, 2006.

[50] S. Corbetta, M. Morandi, M. Novati, M. D. Santambrogio, D. Sciuto, and P. Spoletini, "Internal and External Bitstream Relocation for Partial Dynamic Reconfiguration," *IEEE Transactions on Very Large Scale Integration (VLSI),* vol. 17, no.11, pp. 1650-1654, 2009.

[51] Xilinx, "MicroBlaze Processor Reference Guide," *Xilinx Document: UG081,* 2012.

[52] Xilinx, "PowerPC 405 Processor Block Reference Guide," *Xilinx Document: UG018,* 2010.
[53] Xilinx, "High Performance Computing Using FPGAs," *Xilinx White Paper: WP375,* 2010.
[54] Xilinx, "Zynq-7000 All Programmable SoC," *Xilinx Document: UG585,* 2014.
[55] Altera. (2014). *Altera's User-Customizable SoC [Online]*. Available: http://www.altera.com/devices/processor/soc-fpga/overview/proc-soc-fpga.html
[56] H.T.Kung, "Why Systolic Architectures?," *Computer,* vol. 15, no.1, pp. 37-46, 1982.
[57] K. T. Johnson, A. R. Hurson, and B. Shirazi, "General-Purpose Systolic Arrays " *Computer,* vol. 26, no. 11, pp. 20-31, 1993.
[58] X. Guo, H. Wang, and V. Devabhaktuni, "A Systolic Array-Based FPGA Parallel Architecture for the BLAST Algorithm," *ISRN Bioinformatics,* vol. 2012, pp. 1-11, 2012.
[59] K. Benkrid, Y. Liu, and A. Benkrid, "High Performance Biosequence Database Scanning using FPGAs " *The IEEE International Conference on Acoustics, Speech and Signal Processing,* pp. 361-364, 2007.
[60] P. K. Meher, S. Chandrasekaran, and A. Amira, "FPGA Realization of FIR Filters by Efficient and Flexible Systolization Using Distributed Arithmetic," *IEEE Transactions on Signals Processing,* vol. 56, no. 7, pp. 3009-3017, 2008.
[61] X. Wang and M. Leeser, "A Truly Two-Dimensional Systolic Array FPGA Implementation of QR Decomposition," *ACM Transactions on Embedded Computing Systems,* vol. 9, no. 1, pp. 1-17, 2009.
[62] Z. K. Baker and V. K. Prasanna, "Efficient Hardware Data Mining with the Apriori Algorithm on FPGAs," *The IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM),* pp. 3-12, 2005.
[63] S. Sun and J. Zambreno, "Design and Analysis of a Reconfigurable Platform for Frequent Pattern Mining," *IEEE Transactions on Parallel and Distributed Systems,* vol. 22, no. 9, pp. 1497-1505, 2011.
[64] B. Buyukkurt and W. A. Najjar, "Compiler Generated Systolic Arrays for Wavefront Algorithm Acceleration on FPGAS," *The International Conference on Field Programmable Logic and Applications (FPL),* pp. 655 - 658, 2008.
[65] J. Villarreal, A. Park, W. Najjar, and R. Halstead, "Designing Modular Hardware Accelerators in C with ROCCC 2.0," *The IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM),* pp. 127-134, 2010.
[66] T. Smith and M. Waterman, "Identification of Common Molecular Subsequences," *Journal of Molecular Biology,* vol. 147, no. 1, pp. 195-197, 1981.
[67] G. Weisz and J. C. Hoe, "C-to-CoRAM: Compiling Perfect Loop Nests to the Portable CoRAM Abstraction," *The ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA),* pp. 221-230, 2013.
[68] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. Anderson*, et al.*, "LegUp: High-Level Synthesis for FPGA-Based Processor/Accelerator Systems," *The ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA),* pp. 33-36, 2011.
[69] A. Dasu and R. C. Barnes, "Dynamically Reconfigurable Systolic Array Accelorators," *U.S. Patent 20110264888 A1,* 2011.
[70] A. Otero, Y. E. Krasteva, E. d. l. Torre, and T. Riesgo, "Generic Systolic Array for Run-Time Scalable Cores," *Reconfigurable Computing: Architectures, Tools, and Applications (ARC),* pp. 4-16, 2010.
[71] D. Koch and J. Tørresen, "Advances and Trends in Dynamic Partial Run-Time Reconfiguration," *Dagstuhl Seminar Proceedings 10281-Dynamically Reconfigurable Architectures,* pp. 1-9, 2010.

[72] H. Walder, "Operating System Design for Partially Reconfigurable Logic Devices," *PhD Thesis : Swiss Federal Institute of Technology Zurich,* 2005.

[73] G. Wigley and D. Kearney, "Research Issues in Operating Systems for Reconfigurable Computing," *The International Conference on Engineering of Reconfigurable Systems and Algorithms,* 2002.

[74] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast Template Placement for Reconfigurable Computing Systems," *IEEE Transactions on Design and Test of Computers,* vol. 17, no.1, pp. 68-83, 2000.

[75] H. Walder, C. Steiger, and M. Platzner, "Fast Online Task Placement on FPGAs: Free Space Partitioning and 2D-Hashing," *The International Parallel and Distributed Processing Symposium,* 2003.

[76] M. Morandi, M. Novati, M. D. Santambrogio, and D. Sciuto, "Core Allocation and Relocation Management for a Self Dynamically Reconfigurable Architecture," *The IEEE Computer Society Annual Symposium on VLSI (ISVLSI),* pp. 286-291, 2008.

[77] J. Tabero, J. Septién, H. Mecha, and D. Mozos, "A Low Fragmentation Heuristic for Task Placement in 2D RTR HW Management," *The International Conference on Field Programmable Logic and Applications (FPL),* pp. 241-250, 2004.

[78] Ahmadinia, B. A., C, M. Bednara, and J. Teich, "A New Approach for On-Line Placement on Reconfigurable Devices," *The IEEE International Symposium on Parallel and Distributed Processing,* p. 134, 2004.

[79] J. Y. T. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks," *Performance Evaluation,* vol. 2, no. 4, pp. 237-250, 1982.

[80] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of ACM,* vol. 20, no. 1, pp. 46-61, 1973.

[81] F. Dittmann and S. Frank, "Hard Real-Time Reconfiguration Port Scheduling," *The Conference and Exhibition of Design, Automation & Test in Europe,* pp. 1-6, 2007.

[82] Yi Lu, T. Marconi, K. Bertels, and G. Gaydadjiev, "Online Task Scheduling for the FPGA-Based Partially Reconfigurable Systems," *Reconfigurable Computing: Architectures, Tools, and Applications (ARC),* pp. 216-230, 2009.

[83] Y. Lu, T. Marconi, K. Bertels, and G. Gaydadjiev, "A Communication Aware Online Task Scheduling Algorithm for FPGA-Based Partially Reconfigurable Systems," *The IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM),* pp. 65-68, 2010.

[84] D. Göhringer, M. Hübner, E. Nguepi Zeutebouo, and J. Becker, "Operating System for Runtime Reconfigurable Multiprocessor Systems," *International Journal of Reconfigurable Computing,* vol. 2011, no. 121353, pp. 1-16, 2011.

[85] C. Hong, K. Benkrid, X. Iturbe, A. T. Erdogan, and T. Arslan, "An FPGA Task Allocator with Preliminary First-Fit 2D Packing Algorithms," *The NASA/ESA Conference on Adaptive Hardware and Systems (AHS),* pp. 264-270, 2011.

[86] Xilinx, "LogiCORE IP XPS HWICAP," *Xilinx Document: DS586,* 2010.

[87] Xilinx, "LogiCORE IP AXI HWICAP," *Xilinx Document: PG134,* 2013.

[88] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time Partial Reconfiguration Speed Investigation and Architectural Design Space Exploration," *The International Conference on Field Programmable Logic and Applications (FPL),* pp. 498-502, 2009.

[89] K. Vipin and S. A. Fahmy, "A High Speed Open Source Controller for FPGA Partial Reconfiguration," *The IEEE International Conference on Field-Programmable Technology (FPT),* pp. 61-66, 2012.

[90] M. Hubner, D. Gohringer, J. Noguera, and J. Becker, "Fast Dynamic and Partial Reconfiguration Data Path with Low Hardware Overhead on Xilinx FPGAs," *The IEEE International Symposium on Parallel & Distributed Processing,* pp. 1-8, 2010.

[91] M. Platzner and N. Wehn, "Auto Vision," *in Dynamically Reconfigurable Systems: Architectures, Design Methods and Applications, Springer Science & Business Media,* pp. 388-389, 2010.

[92] S. G. Hansen, D. Koch, and J. Torresen, "High Speed Partial Run-Time Reconfiguration Using Enhanced ICAP Hard Macro," *The IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW),* pp. 174-80, 2011.

[93] D. Koch, C. Beckhoff, and J. Teich, "Bitstream Decompression for High Speed FPGA Configuration from Slow Memories," *The IEEE International Conference on Field-Programmable Technology (FPT),* pp. 161-168, 2007.

[94] S. Liu, R. N. Pittman, and A. Forin, "Minimizing Partial Reconfiguration Overhead with Fully Streaming DMA Engines and Intelligent ICAP Controller " *Microsoft Technical Report: MSR-TR-2009-150,* 2009.

[95] C. Claus, F. H. Müller, and W. Stechele, "Combitgen: A New Approach for Creating Partial Bitstreams in Virtex-II Prodevices," *Workshop on Reconfigurable Computing Proceedings (ARCS 06),* pp. 122-131, 2006.

[96] M. Shelburne, C. Patterson, P. Athanas, M. Jones, B. Martin, and R. Fong, "Metawire: Using FPGA Configuration Circuitry to Emulate a Network-on-Chip," *The International Conference on Field Programmable Logic and Applications (FPL),* pp. 257-262, 2008.

[97] F. Duhem, F. Muller, and P. Lorenzini, "FaRM: Fast Reconfiguration Manager for Reducing Reconfiguration Time Overhead on FPGA," *Reconfigurable Computing: Architectures, Tools, and Applications (ARC),* pp. 253-260, 2011.

[98] J. C. Hoffman and M. S. Pattichis, "A High-Speed Dynamic Partial Reconfiguration Controller Using Direct Memory Access Through a Multiport Memory Controller and Overclocking with Active Feedback," *International Journal of Reconfigurable Computing,* vol. 2011, no. 439072, pp. 1-10, 2011.

[99] Z. Li and S. Hauck, "Configuration Prefetching Techniques for Partial Reconfigurable Coprocessor with Relocation and Defragmentation," *The ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA),* pp. 187-195, 2002.

[100] C. Constantinescu, "Trends and Challenges in VLSI Circuit Reliability," *IEEE Micro,* vol. 23, no. 4, pp. 14-19, 2003.

[101] C. Guérin, V. Huard, and A. Bravaix, "The Energy-Driven Hot-Carrier Degradation Modes of nMOSFETs," *IEEE Transactions on Device and Materials Reliability,* vol. 7, no. 2, pp. 225-235, 2007.

[102] D. Esseni, J. D. Bude, and L. Selmi, "On Interface and Oxide Degradation in VLSI MOSFETs—Part I: Deuterium Effect in CHE Stress Regime," *IEEE Transactions on Electorn Devices,* vol. 49, no. 2, pp. 247-253, 2002.

[103] D. Esseni, J. D. Bude, and L. Selmi, "On Interface and Oxide Degradation in VLSI MOSFETs—Part II: Fowler–Nordheim Stress Regime," *IEEE Transactions on Electorn Devices,* vol. 49, no. 2, pp. 254-263, 2002.

[104] R. Baumann, "The Impact of Technology Scaling on Soft Error Rate Performance and Limits to the Efficacy of Error Correction," *International Electron Devices Meeting,* pp. 329-332, 2002.

[105] T. Semiconductors. (2004). *Soft Errors in Electronic Memory – A White Paper [Online]*. Available: http://www.tezzaron.com/2004/01/

[106] A. H. Johnston, "Scaling and Technology Issues for Soft Error Rates," *The Annual Research Conference on Reliability,* 2000.

[107] Xilinx, "Device Reliability Report, Fourth Quarter 2013," *Xilinx Documnet: UG116,* 2014.

[108] A. Lesea, S. Drimer, J. J. Fabula, C. Carmichael, and P. Alfke, "The Rosetta Experiment: Atmospheric Soft Error Rate Testing in Differing Technology FPGAs," *IEEE Transactions on Device and Materials Reliability,* vol. 317, no. 3, pp. 317-328, 2005.

[109] H. Quinn, K. Morgan, P. Graham, J. Krone, M. Caffrey, and K. Lundgreen, "Domain Crossing Errors: Limitations on Single Device Triple-Modular Redundancy Circuits in Xilinx FPGAs," *IEEE Transactions on Nuclear Science,* vol. 54, no.6, pp. 2037-2043, 2007.

[110] P. Adell and G. Allen, "Assessing and Mitigating Radiation Effects in Xilinx FPGAs," *JPL Publication,* 2008.

[111] Xilinx, "LogiCORE IP Soft Error Mitigation Controller V4.0," *Xilinx Documnet: PG036,* 2013.

[112] Xilinx, "Correcting Single-Event Upsets in Virtex-4 FPGA Configuration Memory," *Xilinx Application Note: XAPP1088,* 2009.

[113] J. Heiner, N. Collins, and M. Wirthlin, "Fault Tolerant ICAP Controller for High-Reliable Internal Scrubbing," *The IEEE Aerospace Conference,* pp. 1-10, 2008.

[114] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, and K. A. Label, "Effectiveness of Internal Versus External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA: Design, Test, and Analysis," *IEEE Transactions on Nuclear Science,* vol. 55, no. 4, pp. 2259–2266, 2008.

[115] G. Asadi and M. B. Tahoori, "Soft Error Rate Estimation and Mitigation for SRAM-Based FPGAs," *The ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA),* pp. 149-160, 2005.

[116] Xilinx, "Soft Error Mitigation Using Prioritized Essential Bits," *Xilinx Application Note: XAPP538,* 2012.

[117] X. Iturbe, M. Azkarate, I. Mart´ınez, J. Perez, and A. Astarloa, "A NovelL SEU,MBU And SHE Handling Strategy for XIilinx Virtex-4 FPGAS," *The International Conference on Field Programmable Logic and Applications (FPL),* pp. 569–573, 2009.

[118] J. R. Azambuja, F. Sousa, L. Rosa, and F. L. Kastensmidt, "Evaluating Large Grain TMR and Selective Partial Reconfiguration for Soft Error Mitigation in SRAM-based FPGAs," *The IEEE International On-Line Testing Symposium,* pp. 101-106, 2009.

[119] C. Bolchini, A. Miele, and M. D. Santambrogio, "TMR and Partial Dynamic Reconfiguration to mitigate SEU faults in FPGAs," *The IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems,* pp. 87-95, 2007.

[120] F. Lima and R. Reis, "Designing Fault Tolerant Systems into SRAM-Based FPGAs," *Design Automation Conference,* pp. 650 - 655, 2003.

[121] Xilinx, "The Xilinx Isolation Design Flow for Fault-Tolerant Systems," *Xilinx Application Note: WP412,* 2013.

[122] F. L. Kastensmidt, L. Sterpone, L. Carro, and M. S. Reorda, "On the Optimal Design of Triple Modular Redundancy Logic for SRAM-based FPGAs," *Design, Automation and Test in Europe,* pp. 1290-1295, 2005.

[123] Xilinx, "Triple Module Redundancy Design Techniques for Virtex FPGAs," *Xilinx Application Note: XAPP197,* 2001.

[124] D. P. Montminy, "Relocable Field Programmable Gate Array Bitstreams for Fault Tolerance," *US Patent: US 7-906-984 B1,* 2011.

[125] S.Dhingra, D.Milton, and S.Stroud, "BIST for Logic and Memory Resources in Virtex-4 FPGAs," *The IEEE North Atlantic Test Workshop,* pp. 19-27, 2006.

[126] C. Stroud, S. Konala, P. Chen, and M. Abramovici, "Built-in Self-Test of Logic Blocks in FPGAs (Finally, a Free Lunch: BIST Without Overhead!) " *The VLSI Test Symposium,* pp. 387-392 1996.

[127] S.-K. Lu and C.-Y. Chen, "Fault Detection and Fault Diagnosis Techniques for Lookup table FPGAs," *The Asain Test Symposium,* pp. 236 - 241 2002.

[128] J. Liu and S. Simmons, "BIST-Diagnosis of Interconnect Fault Locations in FPGA's," *The IEEE Canadian Conference on Electrical and Computer Engineering,* pp. 207 - 210 2003.

[129] C. Stroud, S. Wijesuriya, C. Hamilton, and M. Abramovici, "Built-in Self-Test of FPGA Interconnect," *International Test Conference,* pp. 404-411 1998.

[130] M.Abramovici, "BIST-Based Delay-Fault Testing in FPGAs," *On-line Testing Workshop,* pp. 131-134, 2002.

[131] J. M. Emmert, C. E. Stroud, and M. Abramovici, "Online Fault Tolerance for FPGA Logic Blocks," *IEEE Transactions on Very Large Scale Integration (VLSI),* vol. 15, no. 2, pp. 216-226, 2007.

[132] W.-J. Huang and E. J. McCluskey, "Column-Based Precompiled Configuration Techniques for FPGA Fault Tolerance," *The IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM),* pp. 137-146 2001.

[133] D. P. Montminy, R. O. Baldwin, P. D. Williams, and B. E. Mullins, "Using Relocatable Bitstreams for Fault Tolerance," *The NASA/ESA Conference on Adaptive Hardware and Systems (AHS),* pp. 701-708, 2007.

[134] V. Lakamraju and R. Tessier, "Tolerating Operational Faults in Cluster-based FPGAs," *The ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA),* pp. 187-194, 2000.

[135] Xilinx, "PicoBlaze 8-bit Embedded Microcontroller User Guide," *Xilinx Document: UG129,* 2011.

[136] Xilinx, "IEEE 802.3 Cyclic Redundancy Check," *Xilinx Application Note: XAPP209,* 2001.

[137] Xilinx, "LogiCORE IP Multi-Port Memory Controller (MPMC)," *Xilinx Document: DS643,* 2011.

[138] A. Sudarsanam, R. Kallam, and A. Dasu, "PRR-PRR Dynamic Relocation," *IEEE Computer Architecture Letters,* vol. 8, no. 2, pp. 44-47, 2009.

[139] M. Touiza, G. Ochoa-Ruiz, E.-B. Bourennane, and A. Guessoum, "A Novel Methodology for Accelerating Bitstream Relocation in Partially Reconfigurable Systems," *Microprocessors and Microsystems,* vol. 37, no. 3, pp. 358-372, 2013.

[140] H. M. Hussain, K. Benkrid, A. Ebrahim, A. T. Erdogan, and H. Seker, "Novel Dynamic Partial Reconfiguration Implementation of K-Means Clustering on FPGAs: Comparative Results with GPPs and GPUs," *International Journal of Reconfigurable Computing,* vol. 2012, no. 135926, pp. 1-15, 2012.

[141] E. Stott, P. Sedcole, and P. Y. K. Cheung, "Fault Tolerant Methods for Reliability in FPGAs," *The International Conference on Field Programmable Logic and Applications (FPL),* pp. 415-420, 2008.

[142] X. Iturbe, K. Benkrid, C. Hong, A. Ebrahim, T. Arslan, and Imanol Martinez, "Runtime Scheduling, Allocation, and Execution of Real-Time Hardware Tasks onto Xilinx FPGAs Subject to Fault Occurrence," *International Journal of Reconfigurable Computing,* vol. 2013, no. 905057, pp. 1-32, 2013.

[143] C. Hong, K. Benkrid, X. Iturbe, A. Ebrahim, and T. Arslan, "Efficient On-Chip Task Scheduler and Allocator for Reconfigurable Operating Systems," *IEEE Embedded Systems Letters,* vol. 3, no. 3, pp. 85-88, 2011.

[144] X. Iturbe, K. Benkrid, A. Ebrahim, C. Hong, T. Arslan, and I. Martinez, "Snake: An Efficient Strategy for the Reuse of Circuitry and Partial Computation Results in High-Performance Reconfigurable Computing," *The International Conference on Reconfigurable Computing and FPGAs (ReConFig),* pp. 182-189, 2011.

[145] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA-Based Performance Evaluation of The AES Block Cipher Candidate Algorithm Finalists," *IEEE Transactions on Very Large Scale Integration (VLSI),* vol. 9, no. 4, pp. 545-557, 2001.

[146] T. Good and M. Benaissa, "AES on FPGA From The Fastest to The Smallest," *The International Conference on Cryptographic Hardware and Embedded Systems,* pp. 427-440, 2005.

[147] D. Hwang, M. Chaney, S. Karanam, N. Ton, and K. Gaj, "Comparison of FPGA-Targeted Hardware Implementations of eSTREAM Stream Cipher Candidates " *The State of the Art of Stream Ciphers Workshop,* pp. 151-162, 2008.

[148] HelionTechnology, "AES IP Cores for FPGA," *Helion Technology Document* 2014.

[149] CAST, "AES-P: Programmable AES Encryption - Decryption, Core," *CAST Document,* 2010.

[150] T. Wollinger, J. Guajardo, and C. Paar, "Security on FPGAs: State-of-the-Art Implementations and Attacks," *ACM Transactions on Embedded Computing Systems,* vol. 3, no. 3, pp. 534-574, 2004.

[151] Xilinx, "Solving Today's Design Security Concerns," *Xilinx White Paper: WP365,* 2012.

[152] Altera, "An FPGA Design Security Solution Using a Secure Memory Device," *Altera White Paper: WP-01033-1.0,* 2007.

[153] Xilinx, "PRC/EPRC: Data Integrity and Security Controller for Partial Reconfiguration," *Xilinx Application Note: XAPP887,* 2012.

[154] Altera, "Anti-Tamper Capabilities in FPGA Designs," *Altera White Paper: WP-01066-1.0,* 2008.

[155] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *International Advances in Cryptology Conference,* pp. 388-397, 1999.

[156] F. X. Standaert, S. B. Örs, J. J. Quisquater, and B. Preneel, "Power Analysis Attacks Against FPGA Implementations of the DES," *The International Conference on Field Programmable Logic and Application (FPL),* pp. 84-94, 2004.

[157] O. X. Standaert, E. Peeters, G. Rouvroy, and J. J. Quisquater, "An Overview of Power Analysis Attacks Against Field Programmable Gate Arrays," *Proceedings of The IEEE,* vol. 94, no. 2, pp. 383-394, 2006.

[158] A. Moradi, M. Kasper, and C. Paar, "Black-Box Side-Channel Attacks Highlight the Importance of Countermeasures: An Analysis of the Xilinx Virtex-4 and Virtex-5 Bitstream Encryption Mechanism," *The Conference on Topics in Cryptology,* pp. 1-18, 2012.

[159] A. Moradi, A. Barenghi, T. Kasper, and C. Paar, "On the Vulnerability of FPGA Bitstream Encryption Against Power Analysis Attacks: Axtracting Keys from Xilinx Virtex-II FPGAs," *The ACM conference on Computer and communications security,* pp. 111-124, 2011.

[160] A. L. Masle and W. Luk, "Detecting Power Attacks on Reconfigurable Hardware," *The International Conference on Field Programmable Logic and Application (FPL),* pp. 14-19, 2012.

[161] Xilinx, "Security Monitor IP: Anti-Tamper Soft IP Core for Protection of FPGA Designs and Data Assets," *Xilinx Document: PN 1140,* 2012.

[162] A. Le Masle, G. C. T. Chow, and W. Luk, "Constant Power Reconfigurable Computing," *The International Conference on Field Programmable Logic and Application (FPL),* pp. 1-8, 2011.
[163] Cryptography Research, "Protecting FPGAs from Power Analysis," *Cryptography Research Whitepaper,* 2010.
[164] D. Boneh, R. DeMillo, and R. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," *Advances in Cryptology: EUROCRYPT, Springer-Verlag,* pp. 37-51, 1997.
[165] E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," *Advances in Cryptology: CRYPTO, Springer-Verlag,* pp. 513–525, 1997.
[166] G. Canivet, P. Maistri, R. Leveugle, J. Clédière, F. Valette, and M. Renaudin, "Glitch and Laser Fault Attacks onto a Secure AES Implementation on a SRAM-Based FPGA," *Journal of Cryptology,* vol. 24, no.2, pp. 247-268, 2011.
[167] J. J. Quisquater and D. Samyde, "Measures and Counter-Measures for Smart Cards," *The International Conference on Research in Smart Cards: Smart Card Programming and Security,* pp. 200-210, 2001.
[168] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks," *Proceedings of the IEEE,* vol. 94, no.2, pp. 370-382 2006.
[169] D. Karaklajic, J. M. Schmidt, and I. Verbauwhede, "Hardware Designer's Guide to Fault Attacks," *IEEE Transactions on Very Large Scale Integration (VLSI),* vol. 21, no. 12, pp. 2295-2306, 2013.
[170] Xilinx, "Developing Tamper Resistant Designs with Xilinx Virtex-6 and 7 Series FPGAs," *Xilinx Application Note: XAPP1084,* 2013.
[171] V. K. Prasanna and A. Dandalis, "FPGA-Based Cryptography for Internet Security," *Online Symposium for Electronics Engineers,* 2000.
[172] L. Bossuet, M. Grand, L. Gaspar, V. Fischer, and G. Gogniat, "Architectures of Flexible Symmetric Key Crypto Engines—A Survey: From Hardware Coprocessor to Multi-Crypto-Processor System on Chip," *ACM Computer Surveys,* vol. 45, n. 4, p. 32, 2013.
[173] Q.-H. Khuat, D. Chillet, and M. Hubner, "Considering Reconfiguration Overhead in Scheduling of Dependent Tasks on 2D Reconfigurable FPGA," *The NASA/ESA Conference on Adaptive Hardware and Systems (AHS),* pp. 1-8, 2014.
[174] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw*, et al.*, "PRESENT: An Ultra-Lightweight Block Cipher," *Cryptographic Hardware and Embedded Systems,* pp. 450-466, 2007.
[175] G. Krzysztof. *Present - a lightweight block cipher [Online]*. Available: http://opencores.org/project,present
[176] E. Stavinov. (2009). *Parallel Scrambler Generator [Online]*. Available: http://outputlogic.com/?p=179&cpage=1
[177] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining Association Rules Between Sets of Items in Large Databases," *ACM International Conference on Management of Data,* pp. 207-216, 1993.
[178] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *International Conference on Very Large Data Bases,* pp. 487-499, 1994.
[179] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo, "Fast Discovery of Association Rules," *Advances in Knowledge Discovery and Data Mining,* pp. 307-328, 1996.
[180] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining Frequent Patterns without Candidate Generation," *Data Mining and Knowledge Discovery,* vol. 8, no. 1, pp. 53-87, 2004.

[181] Z. Baker and V. Prasanna, "Efficient Hardware Data Mining with the Apriori Algorithm on FPGAs," *The IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM),* pp. 3-12, 2005.

[182] Z. Baker and V. Prasanna, "An Architecture for Efficient Hardware Data Mining Using Reconfigurable Computing Systems," *The IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM),* pp. 67-75, 2006.

[183] W. H. Wen, J. W. Huang, and M. S. Chen, "Hardware-Enhanced Association Rule Mining with Hashing and Pipelining," *IEEE Transactions in Knowledge and Data Engineering,* vol. 20, no. 6, pp. 784-795, 2008.

[184] S.Sun and J. Zambreno, "Mining Association Rules with Systolic Trees," *The International Conference on Field Programmable Logic and Applications (FPL),* pp. 143-148, 2008.

[185] S.Sun and J. Zambreno, "A Reconfigurable Platform for Frequent Pattern Mining," *The International Conference on Reconfigurable Computing and FPGAs (ReConFig),* pp. 55-60, 2008.

[186] R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng, "KDD-Cup 2000 Organizers' Report: Peeling the Onion," *SIGKDD Explorations,* vol. 2, no. 2, pp. 86-98.

[187] D. Reese and T. H.White, "FPGA Configuration Data Scrambling Using Input Multiplexers," *US Patent: US 8-650-409 B1,* 2014.